Mở đầu

MỞ ĐẦU

Đất nước Việt Nam có lợi thế là có bờ biển dài, nhiều sông ngòi, ao hồ nên việc khai thác và nuôi trồng thủy sản đã mở ra triển vọng lớn về việc cung cấp thủy sản cho nhu cầu đời sống nhân dân, cho xuất khẩu và phục vụ cho việc phát triển ngành chăn nuôi gia súc.

Khai thác và thu hoạch tốt nguồn thủy sản phục vụ cho loài người là một vấn đề cực kỳ quan trọng, nhưng kỹ thuật chế biến còn nhiều hạn chế, vì vậy chưa sử dụng được triệt để nguồn lợi quý giá này.

Theo thống kê nguồn động vật thủy sản đang cung cấp cho nhân lọai trên 20% tổng số protein của thực phẩm, đặc biệt ở nhiều nước có thể lên đến 50%. Giá trị và ý nghĩa dinh dưỡng của thịt cá cũng giống như thịt gia súc nghĩa là protein của thịt cá có đầy đủ các lọai axit amin, mà đặc biệt là có đủ các axit amin không thay thế. Thịt cá tươi có mùi vị thơm ngon, dễ tiêu hóa, dễ hấp thu.

Dầu cá ngoài việc cung cấp lipid cho con người, còn có giá trị sinh học rất cao, đặc biệt là các axit béo không no có tác dụng lớn trong việc trao đổi chất của cơ thể. Ngoài ra, lipid của động vật thủy sản là nguồn rất giàu vitamin A và D.

Trong động vật thủy sản còn chứa nhiều nguyên tố vi lượng và vi lượng rất cần thiết cho cơ thể.

Cá và động vật thủy sản được sử dụng để ăn tươi hoặc chế biến thành nhiều sản phẩm khác nhằm cung cấp tức thời hoặc để dự trữ trong thời gian nhất định. Tuy nhiên, nguyên liệu thủy sản rất dễ ươn hỏng, vì vậy công việc bảo quản phải được đặt lên hàng đầu của khâu chất lượng. Một khi nguyên liệu đã giảm chất lượng thì không có kỹ thuật nào có thể nâng cao chất lượng được.

Nhu cầu tiêu thụ của nhân dân ngày càng cao, vì vậy việc nghiên cứu chế biến ra các sản phẩm mới, hoàn thiện các sản phẩm đang sản xuất để nâng cao chất lượng của sản phẩm là nhiệm vụ quan trọng của các nhà sản xuất, các kỹ sư ngành công nghệ thực phẩm.

Với nội dung giáo trình này nhàm giúp sinh viên hiểu được thành phần hóa học của nguyên liệu thủy sản có ảnh hưởng đến chất lượng sản phẩm trong quá trình chế biến các sản phẩm lạnh, sản phẩm lạnh đông và các sản phẩm khác chế biến từ nguồn nguyên liệu thủy sản.

Giúp cho sinh viên có thể hiểu rõ các biến đổi của động vật thủy sản sau khi chết như sự tê cứng, sự tự phân giải, biến đổi do vi sinh vật có ảnh hưởng rất lớn đến chất lượng sản phẩm thủy sản. Từ đó sinh viên có thể hiểu rõ việc tìm ra phương pháp đánh bắt, sơ chế, vận chuyển và bảo quản thích hợp là rất cần thiết nhằm hạn chế và kéo dài thời gian xảy ra các biến đổi trên. Sinh viên sẽ biết cách đánh giá và chọn nguyên liệu thích hợp để chế biến một số loại sản phẩm thủy sản khác nhau.

Sinh viên cũng được trang bị một số qui trình công nghệ chế biến sản phẩm thủy sản và cách điều khiển qui trình sản xuất để đảm bảo chất lượng sản phẩm.

Với kiến thức của học phần này, sinh viên có thể ứng dụng trong các nhà máy chế biến sản phẩm sấy khô, xông khói, đặt biệt là trong các nhà máy chế biến lạnh đông thủy sản - thế mạnh của vùng Đồng Bằng Sông Cửu Long.

Kiểu con trỏ
Học xong chương này, sinh viên sẽ nắm được các vấn đề sau: + Khái niệm về kiểu dữ liệu “con trỏ”. + Cách khai báo và cách sử dụng biến kiểu con trỏ. + Mối quan hệ giữa mảng và con trỏ.

GIỚI THIỆU KIỂU DỮ LIỆU CON TRỎ

Các biến chúng ta đã biết và sử dụng trước đây đều là biến có kích thước và kiểu dữ liệu xác định. Người ta gọi các biến kiểu này là biến tĩnh. Khi khai báo biến tĩnh, một lượng ô nhớ cho các biến này sẽ được cấp phát mà không cần biết trong quá trình thực thi chương trình có sử dụng hết lượng ô nhớ này hay không. Mặt khác, các biến tĩnh dạng này sẽ tồn tại trong suốt thời gian thực thi chương trình dù có những biến mà chương trình chỉ sử dụng 1 lần rồi bỏ.

Một số hạn chế có thể gặp phải khi sử dụng các biến tĩnh:

- Cấp phát ô nhớ dư, gây ra lãng phí ô nhớ.
- Cấp phát ô nhớ thiếu, chương trình thực thi bị lỗi.

Để tránh những hạn chế trên, ngôn ngữ C cung cấp cho ta một loại biến đặc biệt gọi là biến động với các đặc điểm sau:

- Chỉ phát sinh trong quá trình thực hiện chương trình chứ không phát sinh lúc bắt đầu chương trình.
- Khi chạy chương trình, kích thước của biến, vùng nhớ và địa chỉ vùng nhớ được cấp phát cho biến có thể thay đổi.
- Sau khi sử dụng xong có thể giải phóng để tiết kiệm chỗ trong bộ nhớ.

Tuy nhiên các biến động không có địa chỉ nhất định nên ta không thể truy cập đến chúng được. Vì thế, ngôn ngữ C lại cung cấp cho ta một loại biến đặc biệt nữa để khắc phục tình trạng này, đó là biến con trỏ (pointer) với các đặc điểm:

- Biến con trỏ không chứa dữ liệu mà chỉ chứa địa chỉ của dữ liệu hay chứa địa chỉ của ô nhớ chứa dữ liệu.
- Kích thước của biến con trỏ không phụ thuộc vào kiểu dữ liệu, luôn có kích thước cố định là 2 byte.

KHAI BÁO VÀ SỬ DỤNG BIẾN CON TRỎ

Khai báo biến con trỏ

Cú pháp: <Kiểu> * <Tên con trỏ>

Ý nghĩa: Khai báo một biến có tên là Tên con trỏ dùng để chứa địa chỉ của các biến có kiểu Kiểu.

Ví dụ 1: Khai báo 2 biến a,b có kiểu int và 2 biến pa, pb là 2 biến con trỏ kiểu int.

```
int a, b, *pa, *pb;
```

Ví dụ 2: Khai báo biến f kiểu float và biến pf là con trỏ float

```
float f, *pf;
```

Ghi chú: Nếu chưa muốn khai báo kiểu dữ liệu mà con trỏ ptr đang chỉ đến, ta sử dụng:

```
void *ptr;
```

Sau đó, nếu ta muốn con trỏ ptr chỉ đến kiểu dữ liệu gì cũng được. Tác dụng của khai báo này là chỉ dành ra 2 bytes trong bộ nhớ để cấp phát cho biến con trỏ ptr.

Các thao tác trên con trỏ

Gán địa chỉ của biến cho biến con trỏ

Toán tử & dùng để định vị con trỏ đến địa chỉ của một biến đang làm việc.

Cú pháp: <Tên biến con trỏ>=&<Tên biến>

Giải thích: Ta gán địa chỉ của biến Tên biến cho con trỏ Tên biến con trỏ.

Ví dụ: Gán địa chỉ của biến a cho con trỏ pa, gán địa chỉ của biến b cho con trỏ pb.

pa=&a; pb=&b;

Lúc này, hình ảnh của các biến trong bộ nhớ được mô tả:

			a	b						Bộ nhớ	
	pa	pb	2 byte	2 byte							

Lưu ý:

Khi gán địa chỉ của biến tĩnh cho con trỏ cần phải lưu ý kiểu dữ liệu của chúng. Ví dụ sau đây không đúng do không tương thích kiểu:

int Bien_Nguyen;

float *Con_Tro_Thuc;

...

Con_Tro_Thuc=&Bien_Nguyen;

Phép gán ở đây là sai vì Con_Tro_Thuc là một con trỏ kiểu float (nó chỉ có thể chứa được địa chỉ của biến kiểu float); trong khi đó, Bien_Nguyen có kiểu int.

Nội dung của ô nhớ con trỏ chỉ tới

Để truy cập đến nội dung của ô nhớ mà con trỏ chỉ tới, ta sử dụng cú pháp:

*<Tên biến con trỏ>

Với cách truy cập này thì *<Tên biến con trỏ> có thể coi là một biến có kiểu được mô tả trong phần khai báo biến con trỏ.

Ví dụ: Ví dụ sau đây cho phép khai báo, gán địa chỉ cũng như lấy nội dung vùng nhớ của biến con trỏ:

int x=100;

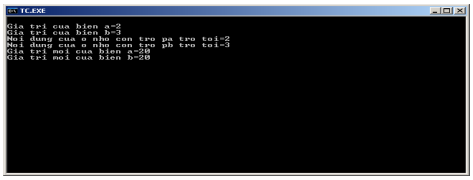
```
int *ptr;

ptr=&x;

int y= *ptr;
```

Lưu ý: Khi gán địa chỉ của một biến cho một biến con trỏ, mọi sự thay đổi trên nội dung ô nhớ con trỏ chỉ tới sẽ làm giá trị của biến thay đổi theo (thực chất nội dung ô nhớ và biến chỉ là một).

Ví dụ: Đoạn chương trình sau thấy rõ sự thay đổi này :

```
#include <stdio.h>

#include <conio.h>

int main()

{

int a,b,*pa,*pb;

a=2;

b=3;

clrscr();

printf("\nGia tri cua bien a=%d \nGia tri cua bien b=%d ",a,b);

pa=&a;

pb=&b;

printf("\nNoi dung cua o nho con tro pa tro toi=%d",*pa);

printf("\nNoi dung cua o nho con tro pb tro toi=%d ",*pb);

*pa=20; /* Thay đổi giá trị của *pa*/

*pb=20; /* Thay đổi giá trị của *pb*/

printf("\nGia tri moi cua bien a=%d \n

Gia tri moi cua bien b=%d ",a,b); /* a, b thay đổi theo*/

getch();

return 0;

}
```

Kết quả thực hiện chương trình:



Cấp phát vùng nhớ cho biến con trỏ

Trước khi sử dụng biến con trỏ, ta nên cấp phát vùng nhớ cho biến con trỏ này quản lý địa chỉ. Việc cấp phát được thực hiện nhờ các hàm malloc(), calloc() trong thư viện alloc.h.

Cú pháp các hàm:

void *malloc(size_t size): Cấp phát vùng nhớ có kích thước là size.

void *calloc(size_t nitems, size_t size): Cấp phát vùng nhớ có kích thước là nitems*size.

Ví dụ: Giả sử ta có khai báo:

int a, *pa, *pb;

pa = (int*)malloc(sizeof(int)); /* Cấp phát vùng nhớ có kích thước bằng với kích thước của một số nguyên */

pb= (int*)calloc(10, sizeof(int)); /* Cấp phát vùng nhớ có thể chứa được 10 số nguyên*/

Lúc này hình ảnh trong bộ nhớ như sau:

								0	1	2	3	4	5	6	7	8	9	
	pa	2 byte			pb				2 byte									

Lưu ý: Khi sử dụng hàm malloc() hay calloc(), ta phải ép kiểu vì nguyên mẫu các hàm này trả về con trỏ kiểu void.

Cấp phát lại vùng nhớ cho biến con trỏ

Trong quá trình thao tác trên biến con trỏ, nếu ta cần cấp phát thêm vùng nhớ có kích thước lớn hơn vùng nhớ đã cấp phát, ta sử dụng hàm realloc().

Cú pháp: void *realloc(void *block, size_t size)

Ý nghĩa:

- Cấp phát lại 1 vùng nhớ cho con trỏ block quản lý, vùng nhớ này có kích thước mới là size; khi cấp phát lại thì nội dung vùng nhớ trước đó vẫn tồn tại.

- Kết quả trả về của hàm là địa chỉ đầu tiên của vùng nhớ mới. Địa chỉ này có thể khác với địa chỉ được chỉ ra khi cấp phát ban đầu.

Ví dụ: Trong ví dụ trên ta có thể cấp phát lại vùng nhớ do con trỏ pa quản lý như sau:

int a, *pa;

pa=(int*)malloc(sizeof(int)); /*Cấp phát vùng nhớ có kích thước 2 byte*/

pa = realloc(pa, 6); /* Cấp phát lại vùng nhớ có kích thước 6 byte*/

Giải phóng vùng nhớ cho biến con trỏ

Một vùng nhớ đã cấp phát cho biến con trỏ, khi không còn sử dụng nữa, ta sẽ thu hồi lại vùng nhớ này nhờ hàm free().

Cú pháp:void free(void *block)

Ý nghĩa: Giải phóng vùng nhớ được quản lý bởi con trỏ block.

Ví dụ: Ở ví dụ trên, sau khi thực hiện xong, ta giải phóng vùng nhớ cho 2 biến con trỏ pa & pb:

free(pa);

free(pb);

Một số phép toán trên con trỏ

a. Phép gán con trỏ: Hai con trỏ cùng kiểu có thể gán cho nhau.

Ví dụ:

int a, *p, *a ; float *f;

a = 5 ; p = &a ; q = p ; /* đúng */

f = p ; /* sai do khác kiểu */

Ta cũng có thể ép kiểu con trỏ theo cú pháp:

(<Kiểu kết quả>*)<Tên con trỏ>

Chẳng hạn, ví dụ trên được viết lại:

int a, *p, *a ; float *f;

a = 5 ; p = &a ; q = p ; /* đúng */

f = (float*)p; /* Đúng nhờ ép kiểu*/

b. Cộng, trừ con trỏ với một số nguyên

Ta có thể cộng (+), trừ (-) 1 con trỏ với 1 số nguyên N nào đó; kết quả trả về là 1 con trỏ. Con trỏ này chỉ đến vùng nhớ cách vùng nhớ của con trỏ hiện tại N phần tử.

Ví dụ: Cho đoạn chương trình sau:

int *pa;

pa = (int*) malloc(20); /* Cấp phát vùng nhớ 20 byte=10 số nguyên*/

int *pb, *pc;

pb = pa + 7;

pc = pb - 3;

Lúc này hình ảnh của pa, pb, pc như sau:

				0	1	2	3	4	5	6	7	8	9		
		pa				pc			pb						

c. Con trỏ NULL: là con trỏ không chứa địa chỉ nào cả. Ta có thể gán giá trị NULL cho 1 con trỏ có kiểu bất kỳ.

d. Lưu ý:

- Ta không thể cộng 2 con trỏ với nhau.

- Phép trừ 2 con trỏ cùng kiểu sẽ trả về 1 giá trị nguyên (int). Đây chính là khoảng cách (số phần tử) giữa 2 con trỏ đó. Chẳng hạn, trong ví dụ trên pc-pa=4.

CON TRỎ VÀ MẢNG

Con trỏ và mảng 1 chiều

Giữa mảng và con trỏ có một sự liên hệ rất chặt chẽ. Những phần tử của mảng có thể được xác định bằng chỉ số trong mảng, bên cạnh đó chúng cũng có thể được xác lập qua biến con trỏ.

Truy cập các phần tử mảng theo dạng con trỏ

Ta có các quy tắc sau:

&<Tên mảng>[0] tương đương với <Tên mảng>

&<Tên mảng> [<Vị trí>] tương đương với <Tên mảng> + <Vị trí>

<Tên mảng>[<Vị trí>] tương đương với *(<Tên mảng> + <Vị trí>)

Ví dụ: Cho 1 mảng 1 chiều các số nguyên a có 5 phần tử, truy cập các phần tử theo kiểu mảng và theo kiểu con trỏ.

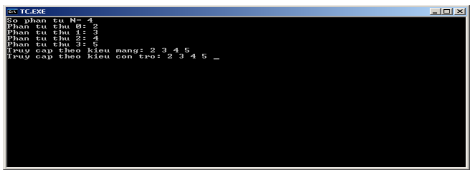
#include <stdio.h>

```
#include <conio.h>
/* Nhập mảng bình thường*/
void NhapMang(int a[], int N){
int i;
for(i=0;i<N;i++)
{
printf("Phan tu thu %d: ",i);scanf("%d",&a[i]);
}
}
/* Nhập mảng theo dạng con trỏ*/
void NhapContro(int a[], int N)
{
int i;
for(i=0;i<N;i++){
printf("Phan tu thu %d: ",i);scanf("%d",a+i);
}
}
int main()
{
int a[20],N,i;
clrscr();
printf("So phan tu N= ");scanf("%d",&N);
NhapMang(a,N); /* NhapContro(a,N)*/
printf("Truy cap theo kieu mang: ");
for(i=0;i<N;i++)
printf("%d ",a[i]);
printf("\nTruy cap theo kieu con tro: ");
for(i=0;i<N;i++)
printf("%d ",*(a+i));
getch();
```

return 0;

}

Kết quả thực thi của chương trình:



Truy xuất từng phần tử đang được quản lý bởi con trỏ theo dạng mảng

<Tên biến>[<Vị trí>] tương đương với *(<Tên biến> + <Vị trí>)

&<Tên biến>[<Vị trí>] tương đương với (<Tên biến> + <Vị trí>)

Trong đó <Tên biến> là biến con trỏ, <Vị trí> là 1 biểu thức số nguyên.

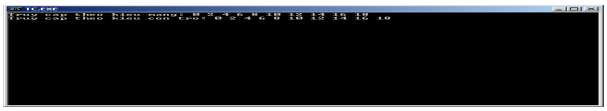
Ví dụ: Giả sử có khai báo:

```
#include <stdio.h>
#include <alloc.h>
#include <conio.h>
int main(){
int *a;
int i;
clrscr();
a=(int*)malloc(sizeof(int)*10);
for(i=0;i<10;i++)
a[i] = 2*i;
printf("Truy cap theo kieu mang: ");
for(i=0;i<10;i++)
printf("%d ",a[i]);
printf("\nTruy cap theo kieu con tro: ");
for(i=0;i<10;i++)
printf("%d ",*(a+i));
getch();
```

return 0;

}

Kết quả chương trình:



Với khai báo ở trên, hình ảnh của con trỏ a trong bộ nhớ:

								0	1	2	3	4	5	6	7	8	9
								0	2	4	6	8	10	12	14	16	18
					a				2 byte								

Con trỏ chỉ đến phần tử mảng

Giả sử con trỏ ptr chỉ đến phần tử a[i] nào đó của mảng a thì:

ptr + j chỉ đến phần tử thứ j sau a[i], tức a[i+j]

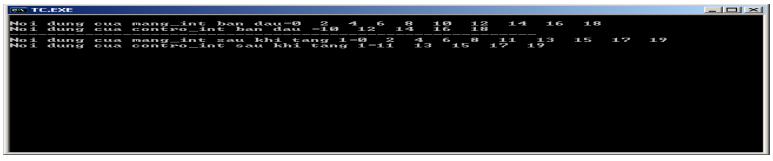
ptr - j chỉ đến phần tử đứng trước a[i], tức a[i-j]

Ví dụ: Giả sử có 1 mảng mang_int, cho con trỏ contro_int chỉ đến phần tử thứ 5 trong mảng. In ra các phần tử của contro_int & mang_int.

```
#include <stdio.h>

#include <conio.h>

#include <alloc.h>

int main()

{

int i,mang_int[10];

int *contro_int;

clrscr();

for(i=0;i<=9;i++)

mang_int[i]=i*2;
```

```
contro_int=&mang_int[5];
printf("\nNoi dung cua mang_int ban dau=");
for (i=0;i<=9;i++)
printf("%d ",mang_int[i]);
printf("\nNoi dung cua contro_int ban dau =");
for (i=0;i<5;i++)
printf("%d ",contro_int[i]);
for(i=0;i<5;i++)
contro_int[i]++;
printf("\n----------");
printf("\nNoi dung cua mang_int sau khi tang 1=");
for (i=0;i<=9;i++)
printf("%d ",mang_int[i]);
printf("\nNoi dung cua contro_int sau khi tang 1=");
for (i=0;i<5;i++)
printf("%d ",contro_int[i]);
if (contro_int!=NULL)
free(contro_int);
getch();
return 0;
}
```

Kết quả chương trình



Con trỏ và mảng nhiều chiều

Ta có thể sử dụng con trỏ thay cho mảng nhiều chiều như sau:

Giả sử ta có mảng 2 chiều và biến con trỏ như sau:

int a[n][m];

int *contro_int;

Thực hiện phép gán contro_int=a;

Khi đó phần tử a[0][0]được quản lý bởi contro_int;

a[0][1]được quản lý bởi contro_int+1;

a[0][2]được quản lý bởi contro_int+2;

...

a[1][0]được quản lý bởi contro_int+m;

a[1][1]được quản lý bởi contro_int+m+1;

...

a[n][m]được quản lý bởi contro_int+n*m;

Tương tự như thế đối với mảng nhiều hơn 2 chiều.

Ví dụ: Sự tương đương giữa mảng 2 chiều và con trỏ.

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
int main()
{
int i,j;
int mang_int[4][5]={1,2,3,4,5,6,7,8,9,10,11,12,13,14,
15,16,17,18,19,20};
int *contro_int;
clrscr();
contro_int=(int*)mang_int;
printf("\nNoi dung cua mang_int ban dau=");
for (i=0;i<4;i++)
{
printf("\n");
for (j=0;j<5;j++)
printf("%d\t",mang_int[i][j]);
```

```
}
printf("\n---------------------------------");
printf("\nNoi dung cua contro_int ban dau \n");
for (i=0;i<20;i++)
printf("%d ",contro_int[i]);
for(i=0;i<20;i++)
contro_int[i]++;
printf("\n---------------------------------------------------------");
printf("\nNoi dung cua mang_int sau khi tang 1=");
for (i=0;i<4;i++)
{
printf("\n");
for (j=0;j<5;j++)
printf("%d\t",mang_int[i][j]);
}
printf("\nNoi dung cua contro_int sau khi tang 1=\n");
for (i=0;i<20;i++)
printf("%d ",contro_int[i]);
if (contro_int!=NULL)
free(contro_int);
getch();
return 0;
}
```

Kết quả thực hiện chương trình như sau:

SORRY, THIS MEDIA TYPE IS NOT SUPPORTED.

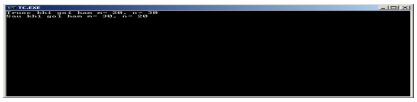
CON TRỎ VÀ THAM SỐ HÌNH THỨC CỦA HÀM

Khi tham số hình thức của hàm là một con trỏ thì theo nguyên tắc gọi hàm ta dùng tham số thực tế là 1 con trỏ có kiểu giống với kiểu của tham số hình thức. Nếu lúc thực thi hàm ta có sự thay đổi trên nội dung vùng nhớ được chỉ bởi con trỏ tham số hình thức thì lúc đó nội dung vùng nhớ được chỉ bởi tham số thực tế cũng sẽ bị thay đổi theo.

Ví dụ : Xét hàm hoán vị được viết như sau :

```
#include<stdio.h>
#include<conio.h>
void HoanVi(int *a, int *b)
{
int c=*a;
*a=*b;
*b=c;
}
int main()
{
int m=20,n=30;
clrscr();
printf("Truoc khi goi ham m= %d, n= %d\n",m,n);
HoanVi(&m,&n);
printf("Sau khi goi ham m= %d, n= %d",m,n);
getch();
return 0;
}
```

Kết quả thực thi chương trình:



m=20n=30&m&nm=20n=30abm=30n=20&m&nm=30n=20&m&nTrước khi gọi hàmKhi gọi hàmSau khi gọi hàm:

a=&m; b= &n; Con trỏ a, b bị giải phóng

Lúc này : *a=m; *b=n; m, n đã thay đổi:

Đổi chỗ ta được :

*a=m=30; *b=n=20;

BÀI TẬP

Mục tiêu

Tiếp cận với một kiểu dữ liệu rất mạnh trong C là kiểu con trỏ. Từ đó, sinh viên có thể xây dựng các ứng dụng bằng cách sử dụng cấp phát động thông qua biến con trỏ.

Nội dung

Thực hiện các bài tập ở chương trước (chương VI : Kiểu mảng) bằng cách sử dụng con trỏ.

Kiểu cấu trúc
Học xong chương này, sinh viên sẽ nắm được các vấn đề sau: + Khái niệm về kiểu cấu trúc. + Cách sử dụng kiểu cấu trúc. + Con trỏ cấu trúc.

KIỂU CẤU TRÚC TRONG C

Khái niệm

Kiểu cấu trúc (Structure) là kiểu dữ liệu bao gồm nhiều thành phần có kiểu khác nhau, mỗi thành phần được gọi là một trường (field)

Sự khác biệt giữa kiểu cấu trúc và kiểu mảng là: các phần tử của mảng là cùng kiểu còn các phần tử của kiểu cấu trúc có thể có kiểu khác nhau.

Hình ảnh của kiểu cấu trúc được minh họa:

1	2	3	4	5	6	Trường7

Đây là cấu trúc có 7 trường

Còn kiểu mảng có dạng:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	Phần tử14

Đây là mảng có 15 phần tử

Định nghĩa kiểu cấu trúc

Cách 1:

struct <Tên cấu trúc>

{

<Kiểu> <Trường 1>;

<Kiểu> <Trường 2>;

……..

<Kiểu> <Trường n>;

};

Cách 2: Sử dụng từ khóa typedef để định nghĩa kiểu:

typedef struct

{

<Kiểu> <Trường 1>;

<Kiểu> <Trường 2>;

……..

<Kiểu> <Trường n>;

} <Tên cấu trúc>;

Trong đó:

- <Tên cấu trúc>: là một tên được đặt theo quy tắc đặt tên của danh biểu; tên này mang ý nghĩa sẽ là tên kiểu cấu trúc.

- <Kiểu> <Trường i> (i=1..n): mỗi trường trong cấu trúc có dữ liệu thuộc kiểu gì (tên của trường phải là một tên được đặt theo quy tắc đặt tên của danh biểu).

Ví dụ 1: Để quản lý ngày, tháng, năm của một ngày trong năm ta có thể khai báo kiểu cấu trúc gồm 3 thông tin: ngày, tháng, năm.

struct NgayThang

{

unsigned char Ngay;

unsigned char Thang;

unsigned int Nam;

};

typedef struct

{

unsigned char Ngay;

unsigned char Thang;

unsigned int Nam;

} NgayThang;

Ví dụ 2: Mỗi sinh viên cần được quản lý bởi các thông tin: mã số sinh viên, họ tên, ngày tháng năm sinh, giới tính, địa chỉ thường trú. Lúc này ta có thể khai báo một struct gồm các thông tin trên.

```
struct SinhVien
{
char MSSV[10];
char HoTen[40];
struct NgayThang NgaySinh;
int Phai;
char DiaChi[40];
};
typedef struct
{
char MSSV[10];
char HoTen[40];
NgayThang NgaySinh;
int Phai;
char DiaChi[40];
} SinhVien;
```

Khai báo biến cấu trúc

Việc khai báo biến cấu trúc cũng tương tự như khai báo biến thuộc kiểu dữ liệu chuẩn.

Cú pháp:

- Đối với cấu trúc được định nghĩa theo cách 1:

struct <Tên cấu trúc> <Biến 1> [, <Biến 2>…];

- Đối với các cấu trúc được định nghĩa theo cách 2:

<Tên cấu trúc> <Biến 1> [, <Biến 2>…];

Ví dụ: Khai báo biến NgaySinh có kiểu cấu trúc NgayThang; biến SV có kiểu cấu trúc SinhVien.

```
struct NgayThang NgaySinh;
```

struct SinhVien SV;

NgayThang NgaySinh;

SinhVien SV;

CÁC THAO TÁC TRÊN BIẾN KIỂU CẤU TRÚC

Truy xuất đến từng trường của biến cấu trúc

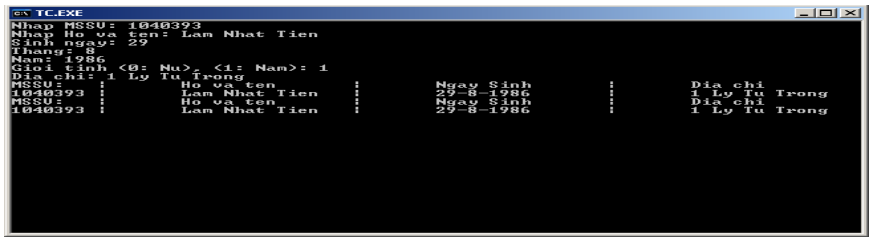
Cú pháp:<Biến cấu trúc>.<Tên trường>

Khi sử dụng cách truy xuất theo kiểu này, các thao tác trên <Biến cấu trúc>.<Tên trường> giống như các thao tác trên các biến của kiểu dữ liệu của <Tên trường>.

Ví dụ : Viết chương trình cho phép đọc dữ liệu từ bàn phím cho biến mẩu tin SinhVien và in biến mẩu tin đó lên màn hình:

```
#include<conio.h>

#include<stdio.h>

#include<string.h>

typedef struct

{

unsigned char Ngay;

unsigned char Thang;

unsigned int Nam;

} NgayThang;

typedef struct

{

char MSSV[10];

char HoTen[40];

NgayThang NgaySinh;

int Phai;

char DiaChi[40];

} SinhVien;
```

```
/* Hàm in lên màn hình 1 mẩu tin SinhVien*/
void InSV(SinhVien s)
{
printf("MSSV: | Ho va ten | Ngay Sinh | Dia chi\n");
printf("%s | %s | %d-%d-%d | %s\n",s.MSSV,s.HoTen,
s.NgaySinh.Ngay,s.NgaySinh.Thang,s.NgaySinh.Nam,s.DiaChi);
}
int main()
{
SinhVien SV, s;
printf("Nhap MSSV: ");gets(SV.MSSV);
printf("Nhap Ho va ten: ");gets(SV.HoTen);
printf("Sinh ngay: ");scanf("%d",&SV.NgaySinh.Ngay);
printf("Thang: ");scanf("%d",&SV.NgaySinh.Thang);
printf("Nam: ");scanf("%d",&SV.NgaySinh.Nam);
printf("Gioi tinh (0: Nu), (1: Nam): ");scanf("%d",&SV.Phai);
flushall();
printf("Dia chi: ");gets(SV.DiaChi);
InSV(SV);
s=SV;/* Gán trị cho mẩu tin s*/
InSV(s);
getch();
return 0;
}
```



Lưu ý:

- Các biến cấu trúc có thể gán cho nhau. Thực chất đây là thao tác trên toàn bộ cấu trúc không phải trên một trường riêng rẽ nào. Chương trình trên dòng s=SV là một ví dụ.

- Với các biến kiểu cấu trúc ta không thể thực hiện được các thao tác sau đây:

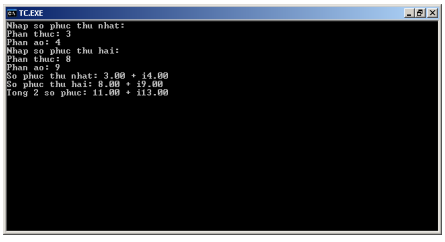
- Sử dụng các hàm xuất nhập trên biến cấu trúc.
- Các phép toán quan hệ, các phép toán số học và logic.

Ví dụ: Nhập vào hai số phức và tính tổng của chúng. Ta biết rằng số phức là một cặp (a,b) trong đó a, b là các số thực, a gọi là phần thực, b là phần ảo. (Đôi khi người ta cũng viết số phức dưới dạng a + ib trong đó i là một đơn vị ảo có tính chất i2=-1). Gọi số phức c1=(a1, b1) và c2=(a2,b2) khi đó tổng của hai số phức c1 và c2 là một số phức c3 mà c3=(a1+a2, b1+b2). Với hiểu biết như vậy ta có thể xem mỗi số phức là một cấu trúc có hai trường, một trường biểu diễn cho phần thực, một trường biểu diễn cho phần ảo. Việc tính tổng của hai số phức được tính bằng cách lấy phần thực cộng với phần thực và phần ảo cộng với phần ảo.

```
#include<conio.h>

#include<stdio.h>

#include<string.h>

typedef struct

{

float Thuc;

float Ao;

} SoPhuc;

/* Hàm in số phức lên màn hình*/

void InSoPhuc(SoPhuc p)

{

printf("%.2f + i%.2f\n",p.Thuc,p.Ao);

}

int main()
```

```
{
SoPhuc p1,p2,p;
clrscr();
printf("Nhap so phuc thu nhat:\n");
printf("Phan thuc: ");scanf("%f",&p1.Thuc);
printf("Phan ao: ");scanf("%f",&p1.Ao);
printf("Nhap so phuc thu hai:\n");
printf("Phan thuc: ");scanf("%f",&p2.Thuc);
printf("Phan ao: ");scanf("%f",&p2.Ao);
printf("So phuc thu nhat: ");
InSoPhuc(p1);
printf("So phuc thu hai: ");
InSoPhuc(p2);
p.Thuc = p1.Thuc+p2.Thuc;
p.Ao = p1.Ao + p2.Ao;
printf("Tong 2 so phuc: ");
InSoPhuc(p);
getch();
return 0;
}
```

Kết quả thực hiện chương trình:



Khởi tạo cấu trúc

Việc khởi tạo cấu trúc có thể được thực hiện trong lúc khai báo biến cấu trúc. Các trường của cấu trúc được khởi tạo được đạt giữa 2 dấu { và }, chúng được phân cách nhau bởi dấu phẩy (,).

Ví dụ: Khởi tạo biến cấu trúc NgaySinh:

struct NgayThang NgaySinh ={29, 8, 1986};

CON TRỎ CẤU TRÚC

Khai báo

Việc khai báo một biến con trỏ kiểu cấu trúc cũng tương tự như khi khai báo một biến con trỏ khác, nghĩa là đặt thêm dấu * vào phía trước tên biến.

Cú pháp: struct <Tên cấu trúc> * <Tên biến con trỏ>;

Ví dụ: Ta có thể khai báo một con trỏ cấu trúc kiểu NgayThang như sau:

struct NgayThang *p;

/* NgayThang *p; // Nếu có định nghĩa kiểu */

Sử dụng các con trỏ kiểu cấu trúc

Khi khai báo biến con trỏ cấu trúc, biến con trỏ chưa có địa chỉ cụ thể. Lúc này nó chỉ mới được cấp phát 2 byte để lưu giữ địa chỉ và được ghi nhận là con trỏ chỉ đến 1 cấu trúc, nhưng chưa chỉ đến 1 đối tượng cụ thể. Muốn thao tác trên con trỏ cấu trúc hợp lệ, cũng tương tự như các con trỏ khác, ta phải:

- Cấp phát một vùng nhớ cho nó (sử dụng hàm malloc() hay calloc)

- Hoặc, cho nó quản lý địa chỉ của một biến cấu trúc nào đó.

Ví dụ: Sau khi khởi tạo giá trị của cấu trúc:

struct NgayThang Ngay = {29,8,1986};

p = &Ngay;

lúc này biến con trỏ p đã chứa địa chỉ của Ngay.

Truy cập các thành phần của cấu trúc đang được quản lý bởi con trỏ

Để truy cập đến từng trường của 1 cấu trúc thông qua con trỏ của nó, ta sử dụng toán tử dấu mũi tên (->: dấu - và dấu >).

Ngoài ra, ta vẫn có thể sử dụng đến phép toán * để truy cập vùng dữ liệu đang được quản lý bởi con trỏ cấu trúc để lấy thông tin cần thiết.

Ví dụ: Sử dụng con trỏ cấu trúc.

```
#include<conio.h>
#include<stdio.h>
typedef struct
{
unsigned char Ngay;
unsigned char Thang;
unsigned int Nam;
} NgayThang;
int main()
{
NgayThang Ng={29,8,1986};
NgayThang *p;
clrscr();
p=&Ng;
printf("Truy cap binh thuong %d-%d-%d\n",
Ng.Ngay,Ng.Thang,Ng.Nam);
printf("Truy cap qua con tro %d-%d-%d\n",
p->Ngay,p->Thang,p->Nam);
printf("Truy cap qua vung nho con tro %d-%d-%d\n",
(*p).Ngay,(*p).Thang,(*p).Nam);
getch();
return 0;
}
```

Kết quả:

BÀI TẬP

Mục đích yêu cầu

Làm quen và biết cách sử dụng kiểu dữ liệu cấu trúc kết hợp với các kiểu dữ liệu đã học. Phân biệt kiểu dữ liệu mảng và kiểu cấu trúc. Thực hiện các bài tập trong phần nội dung.

Nội dung

1. Hãy định nghĩa kiểu:

struct Hoso{

char HoTen[40];

float Diem;

char Loai[10];

};

Viết chương trình nhập vào họ tên, điểm của n học sinh. Xếp loại văn hóa theo cách sau:

ĐiểmXếp loại

9, 10 Giỏi

7, 8 Khá

5, 6 Trung bình

dưới 5 Không đạt

In danh sách lên màn hình theo dạng sau:

XEP LOAI VAN HOA

HO VA TEN DIEM XEPLOAI

Nguyen Van A 7 Kha

Ho Thi B 5 Trung binh

Dang Kim C 4 Khong dat

.....

2. Xem một phân số là một cấu trúc có hai trường là tử số và mẫu số. Hãy viết chương trình thực hiện các phép toán cộng, trừ, nhân, chia hai phân số. (Các kết quả phải tối giản).

3. Tạo một danh sách cán bộ công nhân viên, mỗi người người xem như một cấu trúc bao gồm các trường Ho, Ten, Luong, Tuoi, Dchi. Nhập một số người vào danh sách, sắp xếp tên theo thứ tự từ điển,

in danh sách đã sắp xếp theo mẫu sau:

DANH SACH CAN BO CONG NHAN VIEN

| STT | HO VA TEN | LUONG | TUOI | DIACHI |

| 1 | Nguyen Van A | 333.00 | 26 | Can Tho |

| 2| Dang Kim B | 290.00 | 23 | Vinh Long |

Kiểu dữ liệu

TỔNG QUAN

Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về đối tượng dữ liệu, biến, hằng.
- Khái niệm về kiểu dữ liệu.
- Các phương pháp kiểm tra kiểu và biến đổi kiểu.

Nội dung cốt lõi

- Các khái niệm về đối tượng dữ liệu, kiểu dữ liệu.
- Sự khai báo các đối tượng dữ liệu trong chương trình.
- Kiểm tra kiểu, biến đổi kiểu dữ liệu.
- Vấn đề gán giá trị và khởi tạo biến.

Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản

ĐỐI TƯỢNG DỮ LIỆU

Khái niệm đối tượng dữ liệu

Trong máy tính thực dữ liệu được lưu trữ ở bộ nhớ trong và bộ nhớ ngoài. Trong đó dữ liệu được tổ chức thành các bit, các byte hoặc word. Tuy nhiên trong máy tính ảo của một NNLT nào đó, dữ liệu có tổ chức phức tạp hơn với các mảng, ngăn xếp, số, chuỗi ký tự ...

Người ta sử dụng thuật ngữ đối tượng dữ liệu (ĐTDL) để chỉ một nhóm của một hoặc nhiều mẩu dữ liệu trong máy tính ảo.

Khác với tính chất tĩnh tương đối của các vùng nhớ trong máy tính thực, các ĐTDL và các mối liên hệ nội tại của chúng lại thay đổi một cách động trong quá trình thực hiện chương trình.

Các loại ĐTDL

Xét về mặt cấu trúc thì người ta phân ĐTDL làm hai loại là ĐTDL sơ cấp và ĐTDL có cấu trúc hay cấu trúc dữ liệu.

ĐTDL sơ cấp là một ĐTDL chỉ chứa một giá trị dữ liệu đơn. Hẳng hạn như một số, một kí tự,…

ĐTDL có cấu trúc hay cấu trúc dữ liệu là một tích hợp của các ĐTDL khác. Mỗi ĐTDL thành phần của ĐTDL có cấu trúc được gọi là một phần tử. Mỗi phần tử của cấu trúc dữ liệu có thể là một ĐTDL sơ cấp hay cũng có thể là một ĐTDL có cấu trúc khác. Ví dụ một chuỗi kí tự, một tập hợp các số, một véctơ, một ma trận,…đều là các ĐTDL có cấu trúc.

Xét về mặt nguồn gốc thì có thể phân ĐTDL làm hai loại: ĐTDL tường minh và ĐTDL ẩn.

ĐTDL tường minh là một ĐTDL do người lập trình tạo ra chẳng hạn như các biến, các hằng,… được người lập trình viết ra trong chương trình.

ĐTDL ẩn là một ĐTDL được định nghĩa bởi hệ thống như các ngăn xếp lưu trữ các giá trị trung gian, các mẩu tin kích hoạt chương trình con, các ô nhớ đệm của tập tin... Các ĐTDL này được phát sinh một cách tự động khi cần thiết trong quá trình thực hiện chương trình và người lập trình không thể truy cập đến chúng được.

Thuộc tính của ĐTDL

Thuộc tính của một ĐTDL là một tính chất đặc trưng của ĐTDL đó.

Mỗi ĐTDL có một tập hợp các thuộc tính để phân biệt ĐTDL này với ĐTDL khác.

Các ĐTDL sơ cấp chỉ có một thuộc tính duy nhất là kiểu dữ liệu của đối tượng đó. Các ĐTDL có cấu trúc có thêm các thuộc tính nhằm xác định số lượng, kiểu dữ liệu của các phần tử và các thuộc tính khác.

Giá trị dữ liệu

Giá trị dữ liệu (GTDL) của một ĐTDL sơ cấp có thể là một số, một ký tự hoặc là một giá trị logic tùy thuộc vào kiểu của ĐTDL đó.

Mỗi GTDL thường được biểu diễn bởi một dãy các bit trong bộ nhớ của máy tính.

Cần phân biệt hai khái niệm ĐTDL và GTDL. Một ĐTDL luôn luôn được biểu diễn bởi một khối ô nhớ trong bộ nhớ của máy tính trong khi một GTDL được biểu diễn bởi một dãy các bit. Khi nói rằng một ĐTDL A chứa một GTDL B có nghĩa là: khối ô nhớ biểu diễn cho A chứa dãy bit biểu diễn cho B.

GTDL của một ĐTDL có cấu trúc là một tập hợp các GTDL của các phần tử của ĐTDL có cấu trúc đó.

Thời gian tồn tại

Thời gian tồn tại (lifetime) của một ĐTDL là khoảng thời gian ĐTDL chiếm giữ bộ nhớ của máy tính. Thời gian này được tính từ khi ĐTDL được tạo ra cho đến khi nó bị hủy bỏ trong quá trình thực hiện chương trình.

Các mối liên kết

Một ĐTDL có thể tham gia vào nhiều mối liên kết trong thời gian tồn tại của nó. Các liên kết quan trọng nhất là:

- Sự liên kết của ĐTDL với một hoặc nhiều giá trị. Sự liên kết này có thể bị thay đổi bởi phép gán trị.
- Sự liên kết của một ĐTDL với một hoặc nhiều tên được tham chiếu trong quá trình thực hiện chương trình. Các liên kết này được thiết lập bởi sự khai báo và thay đổi bởi việc gọi và trả chương trình con.
- Sự liên kết của một ĐTDL với một số ĐTDL khác gọi là các hợp thành (component). Các liên kết này thường được biểu diễn bởi giá trị con trỏ và nó có thể bị thay đổi bởi việc thay đổi con trỏ.
- Sự liên kết của một ĐTDL với ô nhớ trong bộ nhớ. Sự liên kết này thường không thể thay đổi một cách trực tiếp bởi người lập trình mà nó được thiết lập và có thể bị thay đổi bởi các thường trình (routine) quản lý bộ nhớ của máy tính ảo.

BIẾN VÀ HẰNG

Biến

Biến là một ĐTDL được người lập trình định nghĩa và đặt tên một cách tường minh trong chương trình. Giá trị của biến có thể bị thay đổi trong thời gian tồn tại của nó.

Tên biến được dùng để xác định và tham khảo tới biến. Trong các NNLT, tên biến thường được quy định dưới dạng một dãy các chữ cái, dấu gạch dưới và các chữ số, bắt đầu bằng một chữ cái và có chiều dài hữu hạn.

Hằng

Hằng là một ĐTDL có tên và giá trị của hằng không thay đổi trong thời gian tồn tại của nó.

Hằng trực kiện (literal constant) là một hằng mà tên của nó là sự mô tả giá trị của nó (chẳng hạn "27" là sự mô tả số thập phân của ĐTDL giá trị 27). Chú ý sự khác biệt giữa 2 giá trị 27. Một cái là một số nguyên được biểu diễn thành một dãy các bit trong bộ nhớ trong quá trình thực hiện chương trình và cái tên "27" là một chuỗi 2 ký tự "2" và "7" mô tả một số nguyên như nó được viết trong chương trình.

KIỂU DỮ LIỆU

Định nghĩa kiểu dữ liệu

Kiểu dữ liệu là một tập hợp các ĐTDL và tập hợp các phép toán thao tác trên các ĐTDL đó.

Mọi NNLT đều xây dựng cho mình một tập các kiểu dữ liệu nguyên thuỷ. Chẳng hạn ngôn ngữ LISP, kiểu dữ liệu chính là các cây nhị phân với các phép toán CAR, CDR và CONS còn đối với các ngôn ngữ cấp cao khác thì các kiểu dữ liệu nguyên thủy thường là: integer, real, character và boolean. Hơn nữa các ngôn ngữ còn cung cấp phương tiện cho phép người lập trình định nghĩa các kiểu dữ liệu mới.

Kiểu dữ liệu trong ngôn ngữ được nghiên cứu trên hai phương diện khác nhau: Sự đặc tả và sự cài đặt kiểu dữ liệu.

Sự đặc tả kiểu dữ liệu

Khi đặc tả một kiểu dữ liệu chúng ta thường quan tâm đến các thành phần cơ bản sau:

- Các thuộc tính nhằm phân biệt các ĐTDL của kiểu.
- Các giá trị mà các ĐTDL của kiểu có thể có.
- Các phép toán có thể thao tác trên các ĐTDL của kiểu.

Ví dụ, xét sự đặc tả kiểu dữ liệu mảng ta thấy:

1.- Các thuộc tính có thể bao gồm: số chiều, miền xác định của chỉ số đối với mỗi chiều và kiểu dữ liệu của các phần tử.

2.- Các giá trị có thể nhận của các phần tử mảng.

3.- Các phép toán có thể bao gồm: phép lựa chọn một phần tử mảng thông qua việc sử dụng chỉ số của phần tử đó, phép gán một mảng cho một mảng khác…

Phép toán

Các phép toán thao tác trên các ĐTDL là một bộ phận không thể thiếu của kiểu dữ liệu. Khi nói đến kiểu dữ liệu mà chúng ta không quan tâm đến các phép toán là chưa hiểu đầy đủ về kiểu dữ liệu đó. Mà dường như khiếm khuyết này lại hay xẩy ra. Ví dụ khi nói đến kiểu integer trong ngôn ngữ Pascal, chúng ta chỉ nghĩ rằng đó là kiểu số nguyên, có các giá trị từ -32768 đến 32767, mà ít khi quan tâm đến các phép toán như +, -, *, … hay nói chính xác hơn chúng ta cứ nghĩ các phép toán này là mặc nhiên phải có. Trong tin học không có cái gì tự nhiên mà có cả, mọi cái hoặc do chúng ta tự tạo ra hoặc sử dụng cái có sẵn do người khác đã tạo ra. Nhấn mạnh việc có mặt các phép toán trong kiểu dữ liệu là để lưu ý chúng ta khi định nghĩa một kiểu dữ liệu mới, phải trang bị cho nó các phép toán cần thiết.

Có hai loại phép toán là các phép toán nguyên thủy được ngôn ngữ định nghĩa và các phép toán do người lập trình định nghĩa như là các chương trình con.

Phép toán trong NNLT về phương diện lôgic là một hàm toán học: đối với một đối số (argument) đã cho nó có một kết quả duy nhất và xác định.

Mỗi một phép toán có một miền xác định (domain) là tập hợp các đối số và một miền giá trị (range) là tập hợp các kết quả có thể tạo ra. Hoạt động của phép toán xác định kết quả được tạo ra đối với tập hợp bất kỳ

các đối số đã cho. Giải thuật chỉ rõ làm thế nào để xác định kết quả đối với tập hợp bất kỳ các đối số đã cho là phương pháp phổ biến để xác định hoạt động của phép toán. Ngoài ra còn có những cách xác định khác chẳng hạn để xác định hoạt động của phép toán nhân chúng ta có thể cho một "bảng nhân" thay vì cho giải thuật của phép nhân hai số.

Để chỉ rõ miền xác định của phép toán, số lượng, thứ tự và kiểu dữ liệu của các đối số, tương tự miền giá trị, số lượng, thứ tự và kiểu dữ liệu của các kết quả người ta thường sử dụng các ký hiệu toán học.

Tên phép toán: Miền xác định -> Miền giá trị

Trong đóMiền xác định = Kiểu đối số X Kiểu đối số X…

(Miền xác định là tập tích Đề-các của các kiểu đối số)

Miền giá trị = Kiểu kết quả X Kiểu kết quả X ...

(Miền giá trị là tập tích Đề-các của các kiểu kết quả)

Khi nghiên cứu các phép toán trên các kiểu dữ liệu chúng ta cần lưu ý các vấn đề sau:

1.- Các phép toán không được xác định đầu vào một cách chắc chắn.

Một phép toán được xác định trên nhiều hơn một miền xác định thường chứa đựng các lỗi. Ví dụ các phép toán số học có thể xác định trên nhiều tập hợp số khác nhau có thể gây ra sự tràn số hoặc một kết quả sai lệch mà ta không thể kiểm soát được.

Ví dụ trong ngôn ngữ Pascal, phép cộng có thể xác định trên nhiều miền xác định khác nhau như integer, real,… nên có thể có những kết quả sai lệch như trong ví dụ sau:

```
var a, b : integer;

begin
```

{1}a:= 32767;

{2}b:= 30000;

{3}writeln(32767+30000);

{4}writeln(a+b);

end.

Kết quả của chương trình trên là 62767 và -2769.

Trong đó 62767 là kết quả của phép cộng 32767+30000. Đây là một kết quả đúng, do máy tính "hiểu" các số 32767 và 30000 là các số thực (real) và phép "+" trong lệnh {3} là "phép cộng các số thực". Ngược lại -2769 là kết quả sai của phép toán a+b. Về mặt toán học thì kết quả của a+b là 62767, nhưng kết quả của chương trình máy tính lại là -2769! Sở dĩ chương trình máy tính (ngôn ngữ Pascal) lại có kết quả này là do hai biến a và b được khai báo là các biến thuộc kiểu integer nên phép "+" trong lệnh {4} được hiểu là "phép cộng các số nguyên". Về nguyên tắc thì tổng a+b phải có giá trị thuộc kiểu integer nhưng do tập giá trị của kiểu integer là các số nguyên từ -32768 đến 32767 nên mới "sinh chuyện".

2.- Các đối số ẩn

Các phép toán trong chương trình thông thường sẽ được gọi với một tập hợp các đối số tường minh (explicit arguments). Tuy nhiên các phép toán có thể truy cập đến những đối số ẩn (implicit arguments) thông qua việc sử dụng các biến toàn cục hoặc tham chiếu các biến không cục bộ khác. Những đối số ẩn như thế sẽ gây khó khăn cho việc kiểm soát giá trị dữ liệu và do đó có thể ảnh hưởng đến kết quả của chương trình.

Ví dụ:

Var x: Integer;

Procedure P;

```
Begin

x:= 0;

End;

Begin

{1} x:=10;

{2}P;

{3}Writeln(x);

End.
```

Trong ví dụ trên, chương trình con P thực hiện việc giá trị 0 cho biến toàn cục x. Trong chương trình chính, mặc dù ta mới gán 10 cho x (lệnh 1), nhưng sau khi gọi thủ tục P (lệnh 2) thì ở lệnh 3, x lại có giá trị 0. Việc chương trình con sử dụng biến không cục bộ như vậy sẽ dễ gây ngộ nhận cho người lập trình rằng x có giá trị 10, đặc biệt khi thủ tục P được định nghĩa ở một đoạn nào đó, xa đoạn chương trình chính.

3.- Hiệu ứng lề

Một phép toán có thể trả về một kết quả ẩn, và các kết quả ẩn như vậy sẽ gây ra hiệu ứng lề (side effect) làm thay đổi giá trị được lưu trữ của các ĐTDL khác mà người lập trình khó lòng kiểm soát. Các phép toán có thể gây nên hiệu ứng lề là phép gán (có trả về một giá trị) và các chương trình con mà tham số được truyền bằng quy chiếu. Chẳng hạn xét ví dụ sau trong Pascal:

```
var m,n: integer;

function f(var a: integer): integer;

begin
```

```
a := 2*a;

f := 5;

end;

begin

m := 10;

n := m + f(m);

writeln(n);

readln;

end.
```

Với mọi số integer a hàm f luôn trả về một kết quả tường minh là 5 và một kết quả ẩn là 2a, chính kết quả ẩn này làm thay đổi giá trị của ĐTDL m do đó n sẽ có giá trị là 25 chứ không phải là 15 như chúng ta lầm tưởng.

Sự cài đặt kiểu dữ liệu

Khi xét sự cài đặt kiểu dữ liệu ta phải quan tâm đến hai yếu tố sau:

- Tổ chức lưu trữ giá trị dữ liệu của kiểu dữ liệu trong bộ nhớ của máy tính hay còn gọi là sự biểu diễn trong bộ nhớ.
- Giải thuật thực hiện các phép toán thao tác trên các giá trị dữ liệu của kiểu.

Hai yếu tố này liên quan chặt chẽ đến nhau, nói chính xác hơn là tuỳ thuộc vào cách thức tổ chức lưu trữ mà có các giải thuật thao tác tương ứng.

SỰ KHAI BÁO

Khái niệm khai báo

Khai báo là một lệnh trong chương trình dùng để chuyển tới bộ dịch, thông tin về số lượng và kiểu của ĐTDL cần thiết trong quá trình thực hiện chương trình.

Nhờ vị trí của khai báo trong chương trình, chẳng hạn đầu chương trình con, sự khai báo có thể chỉ rõ thời gian tồn tại của ĐTDL.

Sự khai báo còn xác định sự liên kết của các ĐTDL với các tên của nó.

Có hai loại khai báo là khai báo tường minh và khai báo ẩn. Khai báo tường minh là sự khai báo do người lập trình viết ra trong chương trình, như trong các khai báo của Pascal. Khai báo ẩn như trong trường hợp các ĐTDL được dùng một cách mặc nhiên mà không cần một sự khai báo tường minh nào. Ví dụ trong ngôn ngữ FORTRAN biến INDEX có thể dùng mà không cần khai báo tường minh và nó được trình biên dịch FORTRAN hiểu một cách mặc nhiên là một biến nguyên bởi vì tên của nó được bắt đầu bởi một trong các chữ cái từ I đến N.

Ngôn ngữ lập trình được chia làm hai loại: ngôn ngữ khai báo, trong đó các ĐTDL phải được khai báo trước khi sử dụng và ngôn ngữ không khai báo, trong đó ĐTDL có thể sử dụng mà không cần phải khai báo. Với ngôn ngữ khai báo, ĐTDL sau khi đã khai báo phải sử dụng đúng như nó đã được khai báo, trong khi đối với ngôn ngữ không khai báo, một ĐTDL có thể sử dụng một cách tuỳ thích. Đây là một trong những lý do làm cho ngôn ngữ không khai báo trở nên mềm dẻo hơn.

Mục đích của sự khai báo

Việc khai báo có các mục đích quan trọng sau:

- Chọn một tổ chức lưu trữ tốt nhất cho ĐTDL. Chẳng hạn trong ngôn ngữ Pascal để lưu trữ ngày trong tháng ta có thể khai báo biến ngay có kiểu là integer được lưu trữ trong bộ nhớ bởi 2 byte. Tuy nhiên trong một tháng chỉ có tối đa 31 ngày nên ta có thể khai báo biến ngay có kiểu miền con 1..31 được lưu trữ trong bộ nhớ chỉ với 1 byte.
- Quản lý bộ nhớ: Sự khai báo cho phép xác định thời gian tồn tại của ĐTDL mà các chương trình quản lý bộ nhớ sử dụng để cấp phát và giải phóng bộ nhớ cho ĐTDL.
- Các phép toán chung. Hầu hết các ngôn ngữ đều dùng các ký hiệu đặc biệt như "+" để chỉ một phép toán nào đó phụ thuộc vào kiểu dữ liệu của đối số. Ví dụ trong Pascal, "A+B" có nghĩa là "phép cộng các số nguyên" nếu A và B thuộc kiểu Integer, "phép cộng các số thực" nếu A và B thuộc kiểu real và là "phép hợp" nếu A và B thuộc kiểu tập hợp. Các phép toán như thế được gọi là các phép toán chung bởi vì nó không chỉ rõ một phép toán nhất định nào. Sự khai báo cho phép bộ dịch xác định một phép toán cụ thể được chỉ định bởi ký hiệu phép toán chung. Ví dụ trong Pascal, từ sự khai báo hai biến A và B, trình biên dịch sẽ xác định được phép toán cụ thể trong ba phép toán, theo đó nếu A, B là các biến integer thì "A+B" là phép cộng hai số nguyên, nếu A, B là hai biến real thì "A+B" là phép cộng hai số thực… Ngược lại trong SNOBOL4 vì không có khai báo kiểu cho biến nên sự xác định phép "+" nào để thực hiện phải được làm tại thời điểm mà một phép "+" bị bắt gặp trong quá trình thực hiện chương trình.
- Kiểm tra kiểu. Mục đích quan trọng nhất của việc khai báo là chúng cho phép kiểm tra kiểu của biến. Vì tính chất quan trọng của việc kiểm tra kiểu nên chúng ta sẽ xem xét nó trong mục sau.

KIỂM TRA KIỂU VÀ BIẾN ĐỔI KIỂU

Khái niệm kiểm tra kiểu

Kiểm tra kiểu là kiểm tra xem kiểu thực nhận được của các đối số trong một phép toán có đúng với kiểu dữ liệu mà các đối số đó cần có hay

không.

Ví dụ trước khi thực hiện lệnh gán X := A * B việc kiểm tra phải được xác định đối với 2 phép toán nhân và phép gán. Trước hết phép nhân phải nhận được 2 tham số A, B có kiểu số, nếu cả A và B đúng là có kiểu số (chẳng hạn số nguyên) thì tiếp tục kiểm tra cho phép toán gán. Tích A*B sẽ là một số nguyên nên X cũng phải là một biến thuộc kiểu nguyên, nếu không đúng như vậy thì có sự sai kiểu.

Kiểm tra kiểu có thể được tiến hành trong lúc chạy chương trình (kiểm tra kiểu động) hoặc trong lúc biên dịch chương trình (kiểm tra kiểu tĩnh).

Kiểm tra kiểu động

Khái niệm:

Kiểm tra kiểu động là kiểm tra kiểu được thực hiện trong khi thực hiện chương trình.

Thông thường kiểm tra kiểu động được thực hiện một cách tức thì trước khi thực hiện một phép toán.

Phương pháp thực hiện:

Để kiểm tra kiểu động người ta phải lưu trữ thông tin về kiểu của mỗi một ĐTDL cùng với ĐTDL đó. Trước khi thực hiện một phép toán thông tin về kiểu của mỗi một đối số được kiểm tra. Nếu kiểu của các đối số là đúng thì phép toán sẽ được thực hiện và kiểu của kết quả sẽ được ghi lại để dùng kiểm tra cho các phép toán sau, ngược lại sẽ có một thông báo lỗi về kiểu .

Ngôn ngữ sử dụng:

Kiểm tra kiểu động được sử dụng trong các ngôn ngữ không khai báo như SNOBOL4, LISP, APL. Trong các ngôn ngữ này không có sự khai báo kiểu cho biến. Kiểu dữ liệu của các biến A và B trong biểu thức "A+B"

có thể thay đổi trong quá trình thực hiện chương trình. Trong những trường hợp như vậy, kiểu của A và B phải được kiểm tra động tại mỗi lần phép cộng được gọi thực hiện. Trong các ngôn ngữ không khai báo, các biến đôi khi được gọi là không định kiểu vì chúng không có kiểu cố định.

Ưu điểm:

Ưu điểm chủ yếu của kiểm tra kiểu động là tính mềm dẻo trong khi viết chương trình: không yêu cầu khai báo kiểu và kiểu của ĐTDL có thể thay đổi trong quá trình thực hiện chương trình. Người lập trình không phải lo lắng về kiểu dữ liệu.

Nhược điểm:

Tuy nhiên kiểm tra kiểu động cũng có một số yếu điểm như sau:

- Có khả năng bỏ sót lỗi về kiểu. Bởi vì việc kiểm tra động chỉ kiểm tra tại thời điểm thực hiện phép toán do đó các phép toán nằm trong nhánh chương trình không được thực hiện thì sẽ không được kiểm tra. Bất kỳ một nhánh chưa được kiểm tra nào đều có thể chứa các đối số có lỗi về kiểu và do đó các lỗi này có thể xuất hiện tại thời điểm sau đó. Ví dụ ta có một đoạn chương trình sau được viết trong một ngôn ngữ kiểm tra kiểu động:

Nhập số a từ bàn phím;

Nhập số b từ bàn phím;

Nếu a > b Thì x := a + b

Ngược lại x := a + “titi”;

Nếu khi thực hiện đoạn chương trình này, người sử dụng luôn luôn nhập số a lớn hơn số b thì điều kiện a>b luôn luôn đúng nên không bao giờ chương trình thực hiện lệnh x := a + “titi” do đó không bao giờ phát hiện lỗi về kiểu: a là một số, không thể cộng với “titi” là một chuỗi.

- Kiểm tra kiểu động đòi hỏi thông tin về kiểu phải được lưu giữ cho mỗi một ĐTDL trong quá trình thực hiện chương trình do đó yêu cầu về bộ nhớ phải lớn.
- Kiểm tra kiểu phải được tiến hành tức thì trước mỗi khi thực hiện một phép toán nên tốc độ thực hiện chương trình chậm.

Kiểm tra kiểu tĩnh

Khái niệm:

Kiểm tra kiểu tĩnh là sự kiểm tra kiểu được thực hiện trong quá trình dịch chương trình.

Phương pháp thực hiện:

Theo nguyên tắc kiểm tra kiểu tĩnh, thông tin về kiểu của ĐTDL phải được cung cấp cho bộ dịch. Thông tin này một phần được cung cấp bởi phép khai báo của người lập trình và một phần bởi ngôn ngữ .

Các thông tin bao gồm:

- Đối với mỗi một phép toán thì đó là số lượng, thứ tự và kiểu dữ liệu của đối số và kiểu của kết quả. Đối với các phép toán nguyên thuỷ thì việc định nghĩa ngôn ngữ sẽ cung cấp các thông tin này còn đối với chương trình con thì người lập trình phải xác định một cách tường minh.
- Đối với mỗi một biến thì đó là kiểu của biến.
- Đối với mỗi một hằng, thì đó là kiểu của đối tượng dữ liệu hằng. Ngữ nghĩa của một hằng trực kiện sẽ chỉ ra kiểu của nó, chẳng hạn "2" là một số nguyên, "2.3" là một số thực.

Kiểm tra kiểu tĩnh được thực hiện như sau: Thông qua đoạn đầu của chương trình, bộ biên dịch tập hợp thông tin từ sự khai báo trong chương trình vào trong bảng danh biểu (symbol table) nơi chứa thông tin về kiểu của các biến và chương trình con. Bộ biên dịch cũng sẽ có thông tin về

các phép toán nguyên thuỷ được định nghĩa bởi ngôn ngữ, các hằng...Khi gặp một phép toán thì phải tra trong bảng danh biểu để xác định kiểu của mỗi một đối số có hợp lệ hay không. Chú ý rằng nếu phép toán là phép toán chung như đã nói ở trên thì có thể có nhiều kiểu hợp lệ cho một đối số. Nếu kiểu của đối số là hợp lệ thì kiểu kết quả được xác định và bộ biên dịch ghi lại thông tin này để kiểm tra các phép toán sau.

Ngôn ngữ sử dụng:

Kiểm tra kiểu tĩnh thường được sử dụng trong các ngôn ngữ khai báo tức là khi viết chương trình, các biến phải được khai báo kiểu trước khi sử dụng như Pascal, C…

Ưu điểm:

- Do phép kiểm tra kiểu tĩnh kiểm tra tất cả các phép toán có thể xuất hiện trong bất kỳ một lệnh nào của chương trình, tất cả các nhánh của chương trình đều được kiểm tra nên không thể có sự sót lỗi vê kiểu.
- Mặt khác thông tin về kiểu không gắn với ĐTDL tại thời điểm thực hiện chương trình nên tiết kiệm được bộ nhớ và tăng tốc độ thực hiện chương trình.

Nhược điểm:

Yếu điểm chủ yếu của kiểm tra kiểu tĩnh là chương trình không mềm dẻo, người lập trình luôn phải lo lắng về việc sử dụng biến không đúng kiểu.

CHUYỂN ĐỔI KIỂU

Trong quá trình kiểm tra kiểu, nếu có sự không tương thích giữa kiểu thực của đối số và kiểu đang được monng đợi của phép toán ấy thì có hai lựa chọn có thể:

- Sự không tương thích kiểu bị báo lỗi hoặc

- Một sự chuyển đổi kiểu tự động được thi hành để đổi kiểu của đối số thực tế thành kiểu đúng với yêu cầu.

Chuyển đổi kiểu là một phép toán được định nghĩa như sau:

Sự chuyển đổi: Kiểu1 -> Kiểu2 nghĩa là sự chuyển đổi lấy ĐTDL của một kiểu và sản sinh ra một ĐTDL "tương ứng" của một kiểu khác. Hầu hết các ngôn ngữ đều cung cấp hai phương pháp chuyển đổi kiểu:

- Trang bị một tập hợp các hàm đã được xây dựng mà người lập trình có thể gọi trong chương trình để tạo ra sự chuyển đổi kiểu. Ví dụ Pascal trang bị hàm ROUND để đổi một ĐTDL số thực thành một đối tượng dữ liệu nguyên với giá trị bằng phần nguyên của số thực.
- Như là một sự chuyển đổi tự động (còn gọi là ép kiểu) do ngôn ngữ thực hiện trong một số trường hợp không tương thích kiểu nào đó. Ví dụ trong Pascal các đối số của phép toán số học "+" có lẫn số thực và số nguyên hoặc khi gán một số nguyên cho một biến số thực thì số nguyên phải được đổi một cách tự động thành kiểu thực.

Đối với kiểm tra kiểu động thì sự chuyển đổi kiểu tự động được diễn ra tại điểm mà sự không tương thích kiểu được tìm thấy trong quá trình thực hiện chương trình. Đối với sự kiểm tra kiểu tĩnh thì một mã phụ sẽ được xen vào trong chương trình đích dùng để gọi tới hàm biến đổi kiểu tại điểm thích hợp trong quá trình thực hiện.

Chuyển đổi kiểu tự động giúp người lập trình khỏi mọi lo lắng về sự sai kiểu và tránh việc gọi tới một số lượng lớn các phép biến đổi kiểu tường minh trong chương trình. Tuy nhiên chúng ta nên tránh việc chuyển đổi kiểu bằng cách viết các phép toán đúng kiểu. Chẳng hạn trong lập trình thay vì viết lệnh x := 1 (với x là biến số thực) ta nên viết x := 1.0, với lệnh trước thì khi thực hiện phải có một sự chuyển đổi kiểu tự động còn với lệnh sau thì không cần nên thời gian thực hiện sẽ nhanh hơn.

GÁN VÀ KHỞI TẠO

Phép gán

Gán trị cho biến là sự lưu trữ giá trị dữ liệu vào trong ô nhớ của biến đó.

Gán trị là một phép toán cơ bản trong các NNLT. Nó dùng để thay đổi sự liên kết của giá trị với ĐTDL.

Nói chung các ngôn ngữ khác nhau thì phép gán cũng khác nhau.

Sự khác nhau đầu tiên là khác nhau về cú pháp, chẳng hạn ta có một số cú pháp lệnh gán như sau:

A := B(Pascal hay Ada)

A = B(C, C++, Fortran, PL/1 và SNOBOL4)

MOVE B TO A(COBOL)

A <- B(APL)

(SETQ A B)(LISP)

Sự khác nhau thứ hai là kết quả trả về của phép gán trị. Nói chung trong các ngôn ngữ, lệnh gán trị không trả về kết quả. Chẳng hạn trong Pascal, đặc tả phép gán là Phép gán (:=) Type1 x Type2 -> Void với sự hoạt động: Đặt giá trị được chứa trong đối tượng dữ liệu Type1 thành bản sao của giá trị được chứa trong đối tượng dữ liệu Type2 và trả về một kết quả có kiểu void (có thể hiểu là không có kết quả trả về).

Trong một số ngôn ngữ như C, C++ và LISP, phép gán trả về trực tiếp một kết quả là một bản sao của giá trị được gán. Chẳng hạn trong C, sự đặc tả phép gán là

Phép gán (=) Type1 x Type2 -> Type3 với sự hoạt động: Đặt giá trị được chứa trong đối tượng dữ liệu Type1 thành bản sao của giá trị được chứa trong đối tượng dữ liệu Type2 và tạo ra một ĐTDL mới Type3 chứa bản sao giá trị của Type2, trả về Type3 như là một kết quả.

Vì phép gán trong Pascal không trả về một kết quả nên chúng ta chỉ sử dụng chức năng “gán trị” của nó mà thôi. Vì không có kết quả trả về nên

mỗi một lệnh ta chỉ có thể viết một phép gán, chẳng hạn để gán giá trị 10 cho hai biến A và B ta phải viết hai lệnh B := 10; A := B; hoặc A := 10; B := 10;

Ngược lại khi lập trình bằng ngôn ngữ C, vì phép gán có trả về một kết quả nên ta có thể viết: A = B = 10; Trong đó phép gán B =10 vừa thực hiện chức năng "gán trị" giá trị 10 cho B vừa trả về 1 giá trị để gán tiếp cho A. Giá trị được trả về như là một kết quả của phép gán B cho A bị bỏ qua vì lệnh không chứa một phép toán nào sau đó nữa.

Phép gán trong ngôn ngữ C++ cũng có cùng cơ chế như trong C, vì vậy khi thiết kế toán tử gán cho một đối tượng nào đó (Overloading toán tử = trong khi xây dựng một lớp nào đó) ta phải viết:

<tên lớp> & operator= (const <tên lớp> & Obj)

{

// Thực hiện việc gán dữ liệu;

return *this;

}

Trong đó tên lớp là tên của lớp chúng ta đang định nghĩa. Phương thức này nhận vào một đối tượng Obj, thực hiện việc gán Obj cho đối tượng hiện hành và trả đối tượng hiện hành này về như một kết quả.

Ví dụ ta tạo một class có tên là point để biểu diễn cho một điểm trong mặt phẳng được đặc trưng bởi hai tọa độ x và y. Trong chương trình ta muốn gán các điểm cho nhau, nên trong khi định nghĩa class point, ta phải định nghĩa toán tử gán. Cụ thể như sau:

class point {

float x;

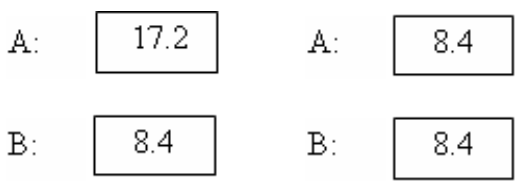
float y;

```
public:

point() {x=0.0 ; y=0.0;} // Phương thức xây dựng mặc nhiên

point (float a, float b) {x=a; y=b;} // Phương thức xây dựng bình thường

point & operator= (const point & p ) // Định nghĩa toán tử gán

{

x = p.x; y = p.y; // Gán dữ liệu

return * this;

}

}; // term
```

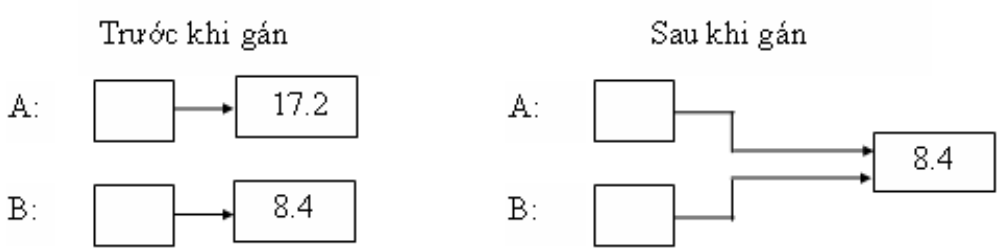
Sự khác nhau cuối cùng của phép gán là ở cách thức tiến hành gán trị. Xét lệnh gán của Pascal "A := B", ở Pascal cũng như một số ngôn ngữ khác, điều này có nghĩa là: "Gán bản sao của giá trị của biến B cho biến A". Bây giờ ta lại xét lệnh gán "A = B" của SNOBOL4. Trong SNOBOL4 thì nó có nghĩa là: "Tạo một biến tên A tham chiếu tới ĐTDL mà B đã tham chiếu". Trong SNOBOL4 cả A và B cùng trỏ tới một ĐTDL.

Pascal A := B (Sao chép ĐTDL khi gán)

Trước Sau



SNOBOL4 A = B (Sao chép sự trỏ đến ĐTDL khi gán)



Cách thực hiện lệnh gán của SNOBOL4 rõ ràng là đã tạo ra một sự lắm tên.

Sự khởi tạo biến

Khởi tạo một biến là gán cho biến đó một giá trị đầu tiên.

Một biến khi được tạo ra thì sẽ được cấp phát ô nhớ nhưng nó vẫn chưa được khởi tạo. Khi nó được gán một giá trị đầu tiên thì mới được khởi tạo.

Các biến chưa được khởi tạo là nguồn gốc của các lỗi lập trình. Khi một biến được cấp phát ô nhớ mà chưa được khởi tạo thì trong ô nhớ của nó cũng có một giá trị ngẫu nhiên nào đó. Thường là một giá trị rác (Khi một ĐTDL nào trước đó đã bị hủy bỏ nhưng giá trị của ĐTDL này trong ô nhớ vẫn còn, giá trị này gọi là giá trị rác). Điều nguy hiểm là giá trị rác này vẫn là một giá trị hợp lệ. Vì thế chương trình có thể xử lý trên giá trị rác này một cách bình thường và chúng ta không thể kiểm sóat được kết quả xử lý đó.

Vì tính chất nghiêm trọng như đã nói trên của biến chưa được khởi tạo, các ngôn ngữ lập trình có thể sử dụng các giải pháp sau để khắc phục:

1.- Nếu biến chưa được khở tạo thì sẽ có giá trị NULL: Khi một biến mới được tạo ra, ô nhớ cấp phát cho nó phải chứa một dãy các bit biểu diễn cho một giá trị “NULL”. Tùy thuộc vào kiểu của biến mà giá trị NULL này sẽ có một giá trị cụ thể, ví dụ nếu là biến số thì NULL là 0, nếu là biến chuỗi kí tự thì NULL là chuỗi rỗng, nếu biến là logic thì NULL là FALSE...

2.- Khởi tạo biến ngay sau khi nó vừa được tạo ra là một cách lập trình tốt và trong một số ngôn ngữ mới đều cung cấp phương tiện để làm điều này một cách dễ dàng. Trong ngôn ngữ Pascal một biến được khởi tạo đồng thời với việc khai báo được gọi là biến có giá trị đầu hay còn gọi là hằng định kiểu.

Ví dụ:

```
const i:integer=10;

a: ARRAY[1..3,1..2] Of Integer = ((11, 12), (21, 22), (31, 32));

var j:integer;

begin

writeln(i); i:= i+1; writeln(i);

for i:=1 to 3 do begin

for j:=1 to 2 do write(a[i,j]:5);

writeln;

end; end.
```

CÂU HỎI ÔN TẬP

1. Xét về mặt cấu trúc thì có các loại đối tượng dữ liệu nào?
2. Thế nào là một đối tượng dữ liệu sơ cấp?
3. Thế nào là một đối tượng dữ liệu có cấu trúc?
4. Đối tượng dữ liệu tường minh là gì?
5. Đối tượng dữ liệu ẩn là gì?
6. Kể tên các mối liên kết của đối tượng dữ liệu.
7. Thế nào là một biến?
8. Thế nào là một hằng?
9. Kiểu dữ liệu là gì?

10. Khi đặc tả một kiểu dữ liệu, chúng ta phải đặc tả những điều gì?
11. Cho ví dụ về một phép toán gây ra hiệu ứng lề.
12. Khi cài đặt một kiểu dữ liệu, chúng ta phải chỉ rõ những điều gì?
13. Mục đích của sự khai báo là gì?
14. Thế nào là kiểm tra kiểu?
15. Khi có sự không tương thích về kiểu thì chương trình dịch phải làm gì?
16. Kể tên các phương pháp kiểm tra kiểu.
17. Kiểm tra kiểu tĩnh được tiến hành trong lúc nào?
18. Kiểm tra kiểu động được tiến hành trong lúc nào?
19. Nêu các điểm mạnh của kiểm tra kiểu tĩnh.
20. Nêu các điểm yếu của kiểm tra kiểu tĩnh.
21. Nêu các điểm mạnh của kiểm tra kiểu động.
22. Nêu các điểm yếu của kiểm tra kiểu động.
23. Thông tin về kiểu trong kiểm tra kiểu tĩnh được lưu trữ ở đâu?
24. Thông tin về kiểu trong kiểm tra kiểu động được lưu trữ ở đâu?
25. Kiểm tra kiểu động được thực hiện trong ngôn ngữ nào?
26. Kiểm tra kiểu tĩnh được thực hiện trong ngôn ngữ nào?
27. Phép gán trị có trả về một kết quả không?
28. Thế nào là khởi tạo một biến?
29. Nếu trong một biểu thức có sử dụng một biến chưa được khởi tạo thì có thể đánh giá (định trị) được biểu thức đó không?

Kiểu dữ liệu có cấu trúc

TỔNG QUAN

Mục tiêu

Sau khi học xong chương này, sinh viên cần phải nắm:

- Khái niệm về kiểu dữ liệu có cấu trúc.
- Đặc tả và phương pháp cài đặt kiểu dữ liệu có cấu trúc.
- Các đặc tả, phương pháp tổ chức lưu trữ, cài đặt các phép toán của một số kiểu dữ liệu có cấu trúc như: vecto, mảng nhiều chiều, mẩu tin, chuỗi ký tự…

Nội dung cốt lõi

- Kiểu dữ liệu có cấu trúc.
- Các đặc tả, phương pháp lưu trữ, hình thức truy xuất, cài đặt các phép toán của một số kiểu dữ liệu có cấu trúc.

Kiến thức cơ bản cần thiết

Kiến thức và kĩ năng lập trình căn bản, kiến thức chương 2.

ĐỊNH NGHĨA KIỂU DỮ LIỆU CÓ CẤU TRÚC

Kiểu dữ liệu có cấu trúc hay còn gọi là cấu trúc dữ liệu (CTDL) là một kiểu dữ liệu mà các ĐTDL của nó là các ĐTDL có cấu trúc.

Như vậy CTDL là một tập hợp các ĐTDL có cấu trúc cùng với tập hợp các phép toán thao tác trên các ĐTDL đó. Các kiểu dữ liệu như mảng, mẩu tin, chuỗi, ngăn xếp (stacks), danh sách, con trỏ, tập hợp và tập tin là các CTDL.

SỰ ĐẶC TẢ KIỂU CẤU TRÚC DỮ LIỆU

Sự đặc tả các thuộc tính

Các thuộc tính chủ yếu của CTDL bao gồm:

Số lượng phần tử

Số lượng các phần tử của một CTDL cho biết kích thước của CTDL, số lượng này có thể cố định hoặc thay đổi tuỳ loại CTDL.

Một CTDL được gọi là có kích thước cố định nếu số lượng các phần tử không thay đổi trong thời gian tồn tại của nó.

Ví dụ các kiểu mảng, mẩu tin là các CTDL có kích thước cố định.

Một CTDL được gọi là có kích thước thay đổi nếu số lượng các phần tử thay đổi một cách động trong thời gian tồn tại của nó.

Ví dụ ngăn xếp, danh sách, tập hợp, chuỗi ký tự và tập tin là các CTDL có kích thước thay đổi. Các phép toán cho phép thêm hoặc bớt các phần tử của cấu trúc làm thay đổi kích thước của cấu trúc.

Kiểu của mỗi một phần tử

Mỗi một phần tử của CTDL có một kiểu dữ liệu nào đó, ta gọi là kiểu phần tử. Kiểu phần tử có thể là một kiểu dữ liệu sơ cấp hoặc một CTDL. Các phần tử trong cùng một CTDL có thể có kiểu phần tử giống nhau hoặc khác nhau.

Một CTDL được gọi là đồng nhất nếu tất cả các phần tử của nó đều có cùng một kiểu.

Ví dụ mảng, chuỗi ký tự, tập hợp và tập tin là các CTDL đồng nhất .

Một CTDL được gọi là không đồng nhất nếu các phần tử của nó có kiểu khác nhau.

Ví dụ mẩu tin là CTDL không đồng nhất.

Tên để dùng cho các phần tử được lựa chọn

Để lựa chọn một phần tử của CTDL cho một xử lý nào đó người ta thường sử dụng một tên. Đối với cấu trúc mảng, tên có thể là một chỉ số nguyên hoặc một dãy chỉ số; đối với mẩu tin, tên là một tên trường. Một số kiểu cấu trúc dữ liệu như ngăn xếp và tập tin cho phép truy nhập đến chỉ một phần tử đặc biệt (phần tử đầu tiên hoặc phần tử hiện hành).

Số lượng lớn nhất các phần tử

Đối với CTDL có kích thước thay đổi như chuỗi ký tự hoặc ngăn xếp, đôi khi người ta quy định thuộc tính kích thước tối đa của cấu trúc để giới hạn số lượng các phần tử của cấu trúc.

Tổ chức cấu trúc

Tổ chức phổ biến nhất là một dãy tuần tự của các phần tử. Vector (mảng một chiều), mẩu tin, chuỗi ký tự, ngăn xếp, danh sách và tập tin là các CTDL có tổ chức kiểu này.

Một số cấu trúc còn được mở rộng thành dạng "nhiều chiều" ví dụ mảng nhiều chiều, mẩu tin mà các phần tử của nó là các mẩu tin, danh sách mà các phần tử của nó là danh sách.

Các phép toán trên cấu trúc dữ liệu

Một số các phép toán đặc thù của CTDL:

Phép toán lựa chọn phần tử của cấu trúc

Phép toán lựa chọn một phần tử là phép toán truy nhập đến một phần tử của CTDL và làm cho nó có thể được xử lý bởi một phép toán khác.

Có hai loại lựa chọn:

Lựa chọn ngẫu nhiên (hay còn gọi là lựa chọn trực tiếp) là sự lựa chọn một phần tử tùy ý của cấu trúc dữ liệu được truy nhập thông qua một cái tên.

Ví dụ để lựa chọn một phần tử nào đó của mảng, ta chỉ ra chỉ số của phần tử đó, để lựa chọn một phần tử của mẩu tin ta sử dụng tên của phần tử đó.

Lựa chọn tuần tự là sự lựa chọn trong đó phần tử được lựa chọn là một phần tử đứng sau các phần tử đã được lựa chọn khác theo tuần tự của việc xử lý hoặc là lựa chọn một phần tử đặc biệt nào đó.

Ví dụ lựa chọn tuần tự các phần tử trong một tập tin hay lựa chọn một phần tử trên đỉnh của ngăn xếp.

Các phép toán thao tác trên toàn bộ cấu trúc dữ liệu

Là các phép toán có thể nhận các CTDL làm các đối số và sản sinh ra kết quả là các CTDL mới. Chẳng hạn phép gán một mẩu tin cho một mẩu tin khác hoặc phép hợp hai tập hợp.

Thêm / bớt các phần tử

Là các phép toán cho phép thêm vào CTDL hoặc loại bỏ khỏi CTDL một số phần tử. Các phép toán này sẽ làm thay đổi số lượng các phần tử trong một CTDL. Việc thêm vào hay loại bỏ một phần tử thường phải chỉ định một vị trí nào đó.

Tạo / hủy CTDL

Là các phép toán tạo ra hoặc xóa bỏ một CTDL.

SỰ CÀI ĐẶT CÁC CẤU TRÚC DỮ LIỆU

Biểu diễn bộ nhớ

Sự biểu diễn bộ nhớ cho một CTDL bao gồm:

- Bộ nhớ cho các phần tử của cấu trúc.

- Bộ mô tả để lưu trữ một số hoặc tất cả các thuộc tính của cấu trúc.

Có hai phương pháp để biểu diễn bộ nhớ là:

Biểu diễn tuần tự

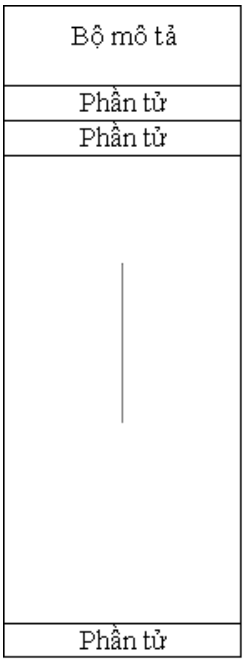
Biểu diễn tuần tự là sự biểu diễn, trong đó CTDL được lưu trữ như một khối các ô nhớ liên tiếp nhau, bắt đầu bằng bộ mô tả sau đó là các phần tử.

Đây là phương pháp được dùng cho các CTDL có kích thước cố định hoặc có kích thước thay đổi nhưng đồng nhất. Chẳng hạn có thể dùng biểu diễn tuần tự để biểu diễn cho mảng, mẩu tin,…

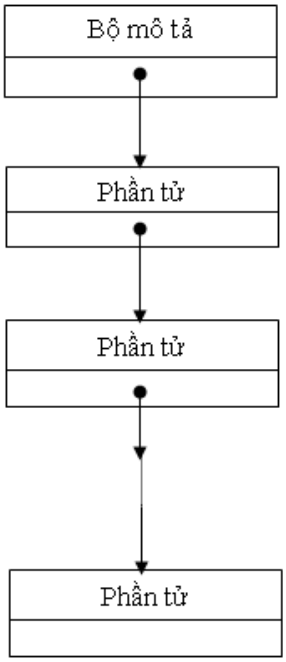
Biểu diễn liên kết

Biểu diễn liên kết là sự biểu diễn, trong đó CTDL được lưu trữ trong nhiều khối ô nhớ tại các vị trí khác nhau trong bộ nhớ, mỗi khối liên kết với khối khác thông qua một con trỏ gọi là con trỏ liên kết.

Phương pháp này thường được sử dụng cho các CTDL có kích thước thay đổi. Chẳng hạn có thể dùng biểu diễn liên kết để biểu diễn cho danh sách, ngăn xếp,…



Biểu diễn tuần tự



Biểu diễn liên kết

Cài đặt các phép toán trên cấu trúc dữ liệu

Phép toán lựa chọn một phần tử là phép toán cơ bản nhất trong các phép toán trên CTDL. Như trên đã trình bày, có hai cách lựa chọn là lựa chọn ngẫu nhiên và lựa chọn tuần tự và hai cách biểu diễn bộ nhớ là biểu diễn tuần tự và biêu diễn liên kết. Vì vậy ở đây chúng ta sẽ xét cách thực hiện mỗi một phương pháp lựa chọn với mỗi một phương pháp biểu diễn bộ nhớ.

Đối với biểu diễn tuần tự

Như trên đã trình bày, trong cách biểu diễn tuần tự, một khối ô nhớ liên tục sẽ được cấp phát để lưu trữ tòan bộ CTDL. Trong đó, vị trí đầu tiên của khối ô nhớ được gọi là địa chỉ cơ sở. Khoảng cách từ địa chỉ cơ sở đến vị trí của phần tử cần lựa chọn được gọi là độ dời của phần tử.

Cách thức truy xuất, được cho bởi tên hoặc chỉ số của phần tử (chẳng hạn chỉ số của một phần tử của mảng), sẽ xác định cách tính độ dời của phần tử như thế nào.

Để lựa chọn ngẫu nhiên một phần tử cần phải xác định vị trí thực của phần tử (tức là địa chỉ của ô nhớ lưu trữ phần tử đó) theo công thức:

Vị trí thực của phần tử = Địa chỉ cơ sở + độ dời của phần tử.

Lựa chọn tuần tự một dãy các phần tử của cấu trúc có thể theo các bước:

- Để chọn phần tử đầu tiên ta dùng cách tính địa chỉ cơ sở cộng với độ dời như đã nói ở trên.

- Đối với các phần tử tiếp theo trong dãy, cộng kích thước của phần tử hiện hành với vị trí của phần tử hiện hành để được vị trí của phần tử kế tiếp.

Đối với biểu diễn liên kết

Như trên đã trình bày, các khối ô nhớ trong biểu diễn liên kết được bố trí rời rạc nhau, khối này nối với khối kia bằng con trỏ và lúc đầu chỉ nắm được con trỏ tới khối đầu tiên. Do đó việc đi đến các khối luôn phải xuất phát từ khối đầu tiên.

Để lựa chọn ngẫu nhiên một phần tử trong cấu trúc liên kết cần phải duyệt một dãy các khối, từ khối đầu tiên đến khối cần lựa chọn.

Lựa chọn tuần tự một dãy các phần tử được thực hiện bằng cách lựa chọn phần tử đầu tiên như đã nói ở trên và sau đó từ phần tử hiện hành, duyệt theo con trỏ để đến phần tử kế tiếp.

VÉCTƠ

Định nghĩa véctơ

Véctơ (còn gọi là mảng một chiều) là một CTDL bao gồm một số cố định các phần tử có kiểu giống nhau được tổ chức thành một dãy tuần tự các phần tử.

Như vậy véctơ là một CTDL có kích thước cố định và đồng nhất.

Sự đặc tả và cú pháp

Đặc tả thuộc tính của véctơ

Các thuộc tính của một véctơ là:

- Số lượng các phần tử, luôn được chỉ rõ bằng cách cho tập chỉ số. Tập chỉ số này thông thường được cho bởi một miền con các số nguyên, trong trường hợp đó, số lượng các phần tử bằng số nguyên cuối cùng - số nguyên đầu tiên + 1. Một cách tổng quát thì tập chỉ số có thể là kiểu liệt kê nào đó, trong trường hợp này, số lượng phần tử bằng số giá trị trong kiểu

liệt kê. Cũng có những ngôn ngữ chỉ định rõ số lượng các phần tử như ngôn ngữ C chẳng hạn.

- Kiểu dữ liệu của mỗi một phần tử, thường được viết rõ trong khai báo.

- Chỉ số được sử dụng để lựa chọn mỗi một phần tử. Nếu tập chỉ số được cho bởi một miền con của tập các số nguyên thì số nguyên đầu tiên chỉ định phần tử đầu tiên số nguyên thứ 2 chỉ định phần tử thứ 2 ...Nếu tập chỉ số là một liệt kê thì giá trị đầu tiên trong liệt kê là chỉ số của phần tử đầu tiên. Nếu ngôn ngữ chỉ định rõ số lượng các phần tử thì 0 là chỉ số của phần tử đầu tiên.

Khai báo véctơ trong Pascal là ARRAY [<tập chỉ số>] OF <kiểu phần tử>.

Ví dụ VAR a: ARRAY[1..10] OF real;

Khai báo này xác định 1 véctơ a có 10 phân tử là các số real. Các phần tử này được lựa chọn bởi các chỉ số từ 1 đến 10.

Miền giá trị của chỉ số không nhất thiết bắt đầu từ 1, ví dụ

Var b: ARRAY [-5..10] OF integer; Với khai báo này thì b là một véctơ có 16 phần tử (10 – (-5) + 1 = 16). Các phần tử được lựa chọn nhờ các chỉ số từ -5 đến 10.

Miền giá trị của chỉ số không nhất thiết là miền con của số nguyên, nó có thể là một liệt kê bất kỳ (hoặc 1 miền con của một liệt kê). Ví dụ:

Type

Ngay = (Chu_nhat, Hai, Ba, Tu, Nam, Sau, Bay);

var

c : ARRAY [Ngay] OF Integer ;

Khai báo này xác đinh véctơ c có 7 phần tử là các số integer, các phần tử của c được lựa chọn nhờ các “chỉ số” từ Chu_nhat đến Bay.

Khai báo véctơ trong ngôn ngữ C là <kiểu phần tử> <tên biến> [<số lượng phần tử>].

Ví dụ int d[10];

Khai báo này xác định véctơ d có 10 phần tử các số int, các phần tử này được lựa chọn nhờ các chỉ số từ 0 đến 9.

Đặc tả các phép toán trên véctơ

Các phép toán trên véctơ bao gồm:

Phép toán lựa chọn một phần tử của véctơ là phép lấy chỉ số, được viết bằng tên của véctơ theo sau là chỉ số của phần tử được lựa chọn đặt trong cặp dấu []. Như vậy phép lựa chọn một phần tử của véctơ là phép lựa chọn trực tiếp.

Ví dụ, với các khai báo trong các ví dụ thuộc phần đặc tả thuộc tính nói trên,

Các phần tử của véctơ a được lựa chọn bằng cách viết a[1], a[2], …, a[10].

Các phần tử của véctơ b được lựa chọn bằng cách viết b[-5], b[-4], …, b[10].

Các phần tử của véctơ c được lựa chọn bằng cách viết c[Chu_nhat], c[Hai], …, c[Bay].

Các phần tử của véctơ d được lựa chọn bằng cách viết d[0], d[1], …, d[9].

Chỉ số có thể là một hằng hoặc một biến (nói chung là một biểu thức), ví dụ a[i] hay a[i+2]. Nhờ chỉ số là một biểu thức nên việc lập trình trở nên đơn giản hơn nhiều nhờ tính khái quát của chỉ số.

Ví dụ để in ra giá trị của 10 phần tử trong véctơ a, thay vì ta phải viết 10 lệnh in các phần tử cụ thể theo kiểu writeln(a[1]); writeln(a[2]); writeln(a[3]); … ta chỉ cần viết một lệnh for i:=1 to 10 do writeln(a[i]);

Các phép toán khác trên véctơ bao gồm các phép toán tạo và hủy bỏ véctơ, gán hai véctơ cho nhau và các phép toán thực hiện như các phép toán số học trên từng cặp 2 véctơ có cùng kích thước. Chẳng hạn phép cộng 2 véctơ (cộng các phần tử tương ứng). Tùy thuộc vào ngôn ngữ mà các phép toán này có hoặc không có.

Cài đặt một véctơ

Biểu diễn bộ nhớ

Biểu diễn bộ nhớ tuần tự được sử dụng để biểu diễn cho một véctơ.

Mô hình sau minh họa cho sự biểu diễn bộ nhớ của véctơ A : ARRAY[LB..UB] OF <kiểu phần tử>.

Địa chỉ cơ sở	Vécto A	Kiểu dữ liệu
	LB	Cận dưới của tập chỉ số
Bộ mô tả	UB	Cận trên của tập chỉ số
	Kiểu phần tử	Kiểu dữ liệu của phần tử
	E	Kích thước mỗi phần tử
	A[LB]	
	A[LB+1]	
Bộ nhớ cho các phần tử của vécto	...	
	A[UB]	

Khối ô nhớ để lưu trữ một véctơ có hai phần: bộ mô tả và bộ nhớ dành cho các phần tử của véctơ. Trong bộ mô tả lưu trữ kiểu dữ liệu của cấu trúc (véctơ A), cận dưới của tập chỉ số (LB - Lower Bound), cận trên của tập chỉ số (UB - Upper Bound), kiểu dữ liệu của phần tử và kích thước mỗi phần tử (E). Bộ nhớ dành cho các phần tử của véctơ lưu trữ liên tiếp các phần tử, từ phần tử đầu tiên (A[LB]) cho đến phần tử cuối cùng (A[UB]). Do các phần tử có cùng một kiểu nên các ô nhớ dành cho các phần tử có kích thước bằng nahu.

Địa chỉ của ô nhớ đầu tiên trong khối gọi là địa chỉ cơ sở.

Giải thuật thực hiện các phép toán

Phép toán lựa chọn một phần tử được thực hiện bằng cách tính vị trí của phần tử cần lựa chọn theo công thức:

Vị trí của phần tử thứ i = + D + (i - LB) * E

Trong đó i là chỉ số của phần tử cần lựa chọn, là địa chỉ cơ sở của khối ô nhớ (địa chỉ word hoặc byte đầu tiên của khối ô nhớ dành cho véctơ) D là kích thước của bộ mô tả, LB là cận dưới của tập chỉ số và E là kích thước của mỗi một đối tượng dữ liệu thành phần (số word hoặc byte cần thiết để lưu trữ một phần tử).

Nếu chỉ số là một giá trị của kiểu liệt kê chứ không phải số nguyên thì hiệu i-LB phải được tính toán một cách thích hợp (chẳng hạn sử dụng hiệu của hai số thứ tự tương ứng của i và LB trong liệt kê).

Phép gán một véctơ cho một véctơ khác có cùng thuộc tính được thực hiện bằng cách sao chép nội dung trong khối ô nhớ biểu diễn véctơ thứ nhất sang khối ô nhớ biểu diễn véctơ thứ hai.

Các phép toán trên toàn bộ véctơ được thực hiện bằng cách sử dụng các vòng lặp xử lý tuần tự các phần tử của véctơ.

MẢNG NHIỀU CHIỀU

Ma trận (mảng hai chiều) được xem như là một véctơ của các véctơ. Một mảng 3 chiều được xem như là một véctơ của các ma trận...

Sự đặc tả và cú pháp

Đặc tả thuộc tính

Mảng nhiều chiều tương tự như véctơ nhưng chỉ có một thuộc tính khác véctơ là mỗi một chiều phải có một tập chỉ số tương ứng.

Chẳng hạn khai báo cho một mảng hai chiều có thể đươc viết dưới dạng

ARRAY[LB1..UB1, LB2..UB2] OF <Kiểu phần tử>

Trong đó tập chỉ số 1 có các giá trị từ LB1 đến UB1, tập chỉ số 2 có các giá trị từ LB2 đến UB2.

Như vậy số lượng các phần tử của mảng hai chiều sẽ là (UB1-LB1+1)*(UB2-LB2+1)

Ví dụ sự khai báo của Pascal:

M= array [1..3, -1..2] of Integer;

Sự khai báo này cho ta thấy mảng M có hai chiều, chiều thứ nhất được xác định bởi tập chỉ số 1..3 và chiều thứ hai được xác định bởi tập chỉ số -1..2. Có thể xem đây là một ma trận có 3 dòng và 4 cột, như vậy sẽ có 12 phần tử, mỗi phần tử có thể lưu trữ một số integer.

Đối với các mảng có số chiều nhiều hơn hai thì cách làm cũng tương tự như mảng hai chiều.

Đặc tả phép toán

Phép lựa chọn một phần tử được thực hiện bằng cách chỉ ra tên mảng và chỉ số của mỗi một chiều.

Chẳng hạn để lựa chọn một phân tử của ma trận ta viết tên ma trận, theo sau là cặp chỉ số dòng, cột phân cách nhau bởi dấu phẩy và đặt trong cặp dấu [], ví dụ M[2,0].

Như vậy phép lựa chọn một phần tử của mảng nhiều chiều là phép lựa chọn trực tiếp.

Sự cài đặt

Sự biểu diễn bộ nhớ

Sự biểu diễn bộ nhớ đối với mảng nhiều chiều tương tự như sự biểu diễn bộ nhớ đối với véctơ. Nghĩa là cũng sử dụng sự biểu diễn tuần tự và khố ô nhớ được chia làm hai phần: bộ mô tả và bộ nhớ cho các phần tử. Bộ mô tả của mảng giống bộ mô tả của véctơ ngoại trừ mỗi một chiều có một cận dưới và cận trên của tập chỉ số của chiều đó. Trong bộ nhớ dành cho các phần tử ta cũng lưu trữ liên tiếp các phần tử theo một trật tự nào đó.

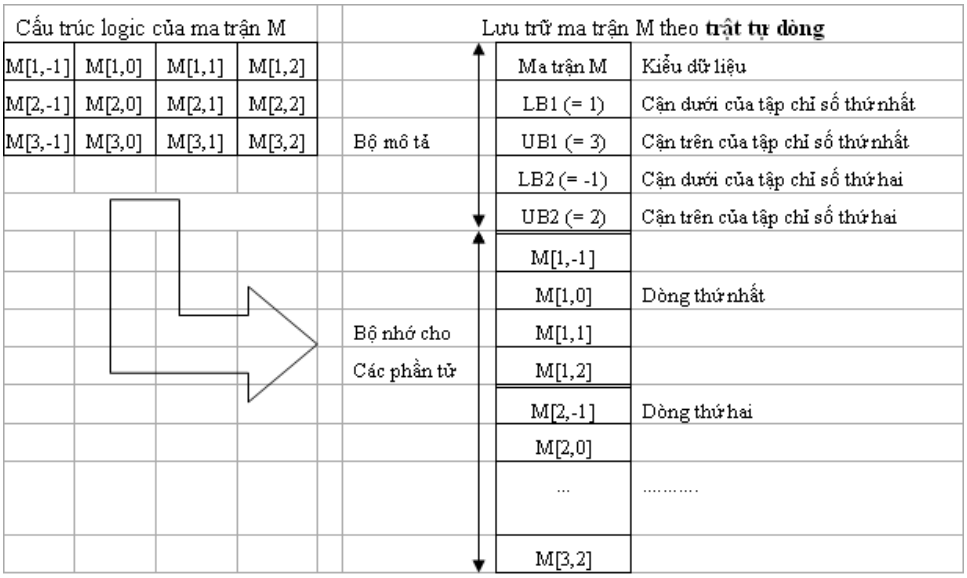
Với ma trận, về mặt logic thì ma trận là một bảng gồm m dòng và n côt, mỗi một ô là một phần tử, nhưng bộ nhớ lại chỉ gồm các ô liên tiếp nhau, vì thế ta phải lưu trữ ma trận theo trật tự dòng hoặc theo trật tự cột.

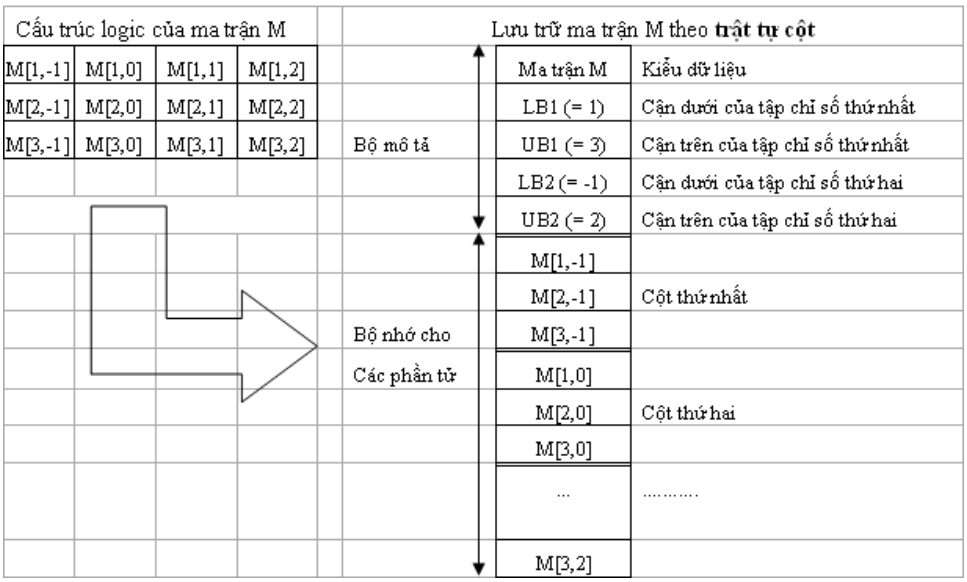
Lưu trữ theo trật tự dòng có nghĩa là trong bộ nhớ dành cho các phần tử ta lưu trữ tuần tự các phần tử trong dòng thứ nhất, tiếp đến là các phần tử trong dòng thứ hai... cho đên dòng cuối cùng.

Lưu trữ theo trật tự cột nghĩa là trong bộ nhớ dành cho các phần tử ta lưu trữ tuần tự các phần tử trong cột thứ nhất, tiếp đến là các phần tử trong cột thứ hai... cho đến cột cuối cùng.

Chẳng hạn với khai báo M: ARRAY [1..3,-1..2] OF Integer; ta có hình ảnh biểu diễn

trong bộ nhớ như các hình sau:





Giải thuật thực hiện phép toán

Để thực hiện phép toán lựa chọn phần tử, ta sử dụng công thức tính vị trí của phần tử trong bộ nhớ.

Với cách lưu trữ theo trật tự dòng của ma trận M, để tính vị trí của M[i,j], đầu tiên ta xác định số dòng cần nhảy qua: (i-LB1) nhân với độ dài của mỗi dòng để xác định vị trí bắt đầu của dòng thứ i và sau đó tìm vị trí thứ J trong dòng này như đối với 1 véctơ. Như vậy, vị trí của phần tử M[i,j] được tính bởi:

Vị trí của M [i,j] = + D + (i-LB1) x S + (j-LB2) x E

Trong đó: là địa chỉ cơ sở.

D là độ lớn của bộ mô tả.

S là độ lớn của mỗi dòng = (UB2 - LB2 +1) x E.

LB1 là cận dưới của chỉ số thứ nhất.

LB2,UB2 tương ứng là cận dưới và cận trên của chỉ số thứ hai.

Tương tự ta có thể thành lập công thức tính vị trí của phần tử M[i,j] trong trường hợp ma trận M được tổ chức lưu trữ theo trật tự cột.

Tổng quát hóa công thức này cho mảng nhiều chiều hơn là một điều đơn giản.

MẨU TIN

Định nghĩa mẩu tin

Mẩu tin là một CTDL bao gồm một số cố định các phần tử có kiểu khác nhau.

Như vậy, mẩu tin là một CTDL có kích thước cố định và không đồng nhất. Các phần tử của mẩu tin được gọi là các trường.

Sự đặc tả và cú pháp

Đặc tả thuộc tính

Các thuộc tính của một mẩu tin phải được chỉ rõ trong phép khai báo, chúng bao gồm:

1. Số lượng các phần tử.

2. Kiểu dữ liệu của các phần tử (Các phần tử có thể có kiểu khác nhau).

3. Mỗi phần tử được cho bởi tên phần tử (tên trường).

Cú pháp khai báo mẩu tin của Pascal:

Nhan_vien: RECORD

Ma: Integer; {Mã nhân viên}

Ho_ten: String[25];

Tuoi: Integer; {Tuổi}

Luong: Real; {Hệ số lương}

END

Việc khai báo này đặc tả một mẩu tin có 4 phần tử của các kiểu Integer, Real và String. Mỗi phần tử có một tên: Ma, Ho_ten, Tuoi và Luong. Để chọn một phần tử của mẩu tin ta sử dụng tên của phần tử (trường) đó, chẳng hạn trong Pascal, Nhan_vien.Luong là để truy xuất tới phần tử Luong của mẩu tin Nhan_vien.

Đặc tả phép toán

Lựa chọn một phần tử là phép toán cơ bản cuả mẩu tin. Phép toán này được thực hiện bằng cách chỉ ra tên trực kiện của phần tử.

Ví dụ để lựa chọn phần tử thứ 4 của mẩu tin Nhan_vien ta viết: Nhan_vien.Luong.

Phép toán lựa chọn một phần tử của mẩu tin là sự lựa chọn trực tiếp.

Mặc dù đều là lựa chọn trực tiếp, nhưng có khác biệt so với cách lựa chọn phần tử của véctơ. Điểm khác biệt ở đây là: đối với véctơ, ta có thể sử dụng giá trị của một biểu thức làm chỉ số, chẳng hạn VECTO[i+1], còn đối với mẩu tin thì bắt buộc phải chỉ rõ tên trực kiện, chứ không thể là biểu thức.

Ngoài phép toán lựa chọn phần tử, phép gán các mẩu tin có cùng cấu trúc là một phép toán phổ biến được các ngôn ngữ đưa vào. Chẳng hạn Nhan_vien := InputRec trong đó InputRec có các thuộc tính giống hệt Nhan_vien.

Sự cài đặt

Biểu diễn bộ nhớ

Biểu diễn bộ nhớ tuần tự được sử dụng để lưu trữ một mẩu tin. Một khối liên tục các ô nhớ được dùng để lưu trữ cho một mẩu tin, trong khối đó, mỗi ô biểu diễn cho một trường. Có thể cũng cần sử dụng bộ mô tả riêng cho từng trường để lưu trữ thuộc tính của các trường đó. Do các trường có kiểu khác nhau nên ô nhớ dành cho chúng cũng có kích thước khác nhau.

Giải thuật thực hiên phép toán

Việc lựa chọn phần tử được thực hiện một cách dễ dàng vì tên trường được biết đến thông qua việc dịch chứ không phải được tính toán thông qua việc thực hiện như đối với véctơ. Việc khai báo mẩu tin còn cho phép xác định kích thước và vị trí của nó trong ô nhớ thông qua việc dịch. Kết quả là độ dời của phần tử bất kỳ có thể được tính thông qua việc dịch.

Chẳng hạn với mẩu tin Nhan_vien, các phần tử của nó được lưu trữ trong bộ nhớ như sau:

22901	Ma
Nguyen Van A	Ho_ten
20	Tuoi

2.18	Luong

Vị trí của một phần tử bất kỳ được tính một cách dễ dàng. Chẳng hạn

Vị trí của Tuoi = + Kích thước của Ma + Kích thước của Ho_ten.

Trong đó là địa chỉ cơ sở của khối ô nhớ biểu diễn cho Nhan_vien.

Phép toán gán toàn bộ một mẩu tin cho một mẩu tin khác có cùng cấu trúc được thực hiện một cách đơn giản là copy nội dung khối ô nhớ biểu diễn cho mẩu tin thứ nhất sang khối ô nhớ biểu diễn cho mẩu tin thứ 2.

MẨU TIN CÓ CẤU TRÚC THAY ĐỔI

Đặc tả và khai báo

Trước hết ta xét ví dụ sau:

Giả sử trong một xí nghiệp có hai loại công nhân là công nhân trong biên chế và công nhân hợp đồng. Đối với công nhân trong biên chế thì lương sẽ được tính bằng số ngày công * mức lương tối thiểu * hệ số /20, những ngày nghỉ bảo hiểm xã hội, họ được trả lương bảo hiểm xã hội. Ngược lại công nhân hợp đồng chỉ được trả lương bằng số ngày công * đơn giá công nhật và họ không được trả lương bảo hiểm xã hội.

Ta thấy, hai loại công nhân này có những thông tin chung là họ tên, số ngày công, tiền lương và loại công nhân (biên chế hay hợp đồng). Mỗi loại công nhân lại có các thông tin riêng. Đối với công nhân trong biên chế, ta cần thêm các thông tin: hệ số lương và số ngày nghỉ bảo hiểm xã hội. Đối với công nhân hợp đồng, ta cần thêm thông tin về đơn giá công nhật.

Nếu sử dụng mẩu tin bình thường để lưu trữ thông tin về hai loại công nhân này, ta cần tất cả 7 trường để lưu trữ 4 thông tin chung và 3 thông tin riêng. Khối ô nhớ cần cấp phát phải đủ để lưu trữ cả 7 trường nhưng việc sử dụng khối ô nhớ lại bị dư, do đối với công nhân biên chế ta chỉ cần 6 trường, đối với công nhân hợp đồng ta chỉ cần 5 trường!

Đặc tả thuộc tính

Để giải quyết vấn đề lãng phí bộ nhớ, trong một số ngôn ngữ lập trình có một loại CTDL gọi là mẩu tin có cấu trúc thay đổi.

Mỗi một cấu trúc sẽ có một số trường giống nhau cho mọi loại mẩu tin và một số trường khác nhau cho từng loại mẩu tin. Các trường giống nhau gọi là phần chung hay phần tĩnh, các trường khác nhau này gọi là phần động hay phần thay đổi của mẩu tin.

Chẳng hạn đối với bài toán nêu trên thì mỗi công nhân được lưu trong một mẩu tin, có các trường thuộc phần chung đó là Ho_Ten, Ngay_Cong, Tien_Luong. Ngoài ra tùy thuộc vào loại công nhân là biên chế hay hợp đồng mà có các trường riêng. Đối với công nhân trong biên chế ta cần thêm các trường He_So và Nghi_Bhxh để lưu trữ hệ số lương và số ngày nghỉ bảo hiểm xã hội. Đối với công nhân hợp đồng ta chỉ cần thêm một trường là Gia_Cong_Nhat để lưu trữ giá công nhật cho mỗi người.

Khai báo trong Pascal như sau:

```
TYPE

loai_cong_nhan = (bien_che,hop_dong);

VAR

Cong_Nhan : RECORD

ho_ten: String[20];

ngay_cong: Real;

luong: Real;

CASE loai: loai_cong_nhan OF

bien_che:

(he_so: Real;

nghi_bhxh:Real);

hop_dong:

(gia_cong_nhat: Real);

END;
```

Khai báo trên định nghĩa một mẩu tin có cấu trúc thay đổi. Mẩu tin luôn luôn có các trường Ho_Ten, Ngay_Cong, Luong và Loai. Khi giá trị của Loai = "bien_che" thì mẩu tin còn có các trường He_So và Nghi_Bhxh, trong khi đó nếu giá trị của Loai = "hop_dong" thì nó lại có trường Gia_Cong_Nhat.

Đặc tả phép toán

Phép toán lựa chọn các phần tử của mẩu tin có cấu trúc thay đổi cũng giống như mẩu tin bình thường. Chẳng hạn ta có thể sử dụng Cong_Nhan.Luong, Cong_Nhan.He_So hay Cong_Nhan.Gia_Cong_Nhat. Tuy nhiên các trường thuộc phần động chỉ tồn tại trong một

thời điểm nhất định do đó khi chúng ta truy xuất tới một tên trường mà nó không tồn tại thì sẽ bị lỗi. Trường Loai trong ví dụ trên là rất quan trọng vì nó chỉ ra phần động nào của mẩu tin được sử dụng trong quá trình thực hiện chương trình. Người đọc có thể tham khảo ví dụ tương đối hoàn chỉnh viết bằng Pascal.

```
uses crt;

Const luong_toi_thieu = 290000;

Type

Loai_cong_nhan = (bien_che, hop_dong);

Cong_nhan = Record

ho_ten : String[20];

Ngay_cong : real;

luong : real;

Case loai: Loai_cong_nhan of

bien_che: (He_so, so_ngay_nghi_BHXH : real);

hop_dong: (don_gia: real);

end;

danh_sach_cong_nhan = Array[1..10] of cong_nhan;

Var

n : integer; ho_so : danh_sach_cong_nhan;

{Nhập danh sách công nhân, và các thông tin liên quan đến lao động}

Procedure Nhap (var ho_so: danh_sach_cong_nhan; var n: integer);

Var

i: integer;

loaicn : char;

Begin

write('So cong nhan: '); readln(n);
```

```
For i:=1 to n do with ho_so[i] do begin
Writeln('Cong nhan ',i);
Write('Ho va Ten: '); readln(ho_ten);
Write('Loai cong nhan: A la bien che, B la hop dong ');
readln(loaicn);
If Upcase(loaicn) ='A' then loai := bien_che else loai := hop_dong;
write('So ngay cong: '); readln(ngay_cong);
if loai = bien_che then begin
write('He so: '); readln(he_so);
write('So ngay nghi bao hiem: '); readln(so_ngay_nghi_BHXH);
end else begin
write('Don gia hop dong: '); readln(don_gia);
end;
end; { with Ho_so[i] }
end; {nhap}
{Tính lương cho từng công nhân, theo công thức của từng loại công nhân}
Procedure Tinh_luong (var ho_so: danh_sach_cong_nhan; n: integer);
Var
i : integer; luong_binh_quan: real;
begin
for i:=1 to n do with ho_so[i] do begin
if loai = bien_che then begin {tính lương của công nhân biên chế}
luong_binh_quan := he_so * luong_toi_thieu/20;
luong := ngay_cong * luong_binh_quan +
so_ngay_nghi_BHXH * luong_binh_quan*0.80;
```

```
end else {tính lương của công nhân hợp đồng}
luong := ngay_cong * don_gia;
end; { with Ho_so[i] }
end; {Tinh_luong }
Procedure In_luong (ho_so: danh_sach_cong_nhan; n: integer);
Var
i : integer;
begin
for i:=1 to n do with ho_so[i] do begin
Write(ho_ten:25);
If loai = bien_che then write('Bien che':10)
else write('Hop dong':10);
write(ngay_cong:5:1);
if loai = bien_che then begin
write(he_so:5:1);
write(so_ngay_nghi_BHXH:5:1);
end else
write(don_gia:10:2);
writeln(luong:10:2);
end; { with Ho_so[i] }
end; { In_luong }
begin {Chuong trinh chinh}
nhap(ho_so,n);
tinh_luong(ho_so,n);
in_luong(ho_so,n);
```

readln;

end.

Cài đặt mẩu tin có cấu trúc thay đổi

Biểu diễn bộ nhớ

Biểu diễn tuần tự sẽ được sử dụng để biểu diễn cho một mẩu tin có cấu trúc thay đổi.

Thông qua việc dịch, tổng bộ nhớ cần để lưu các phần tử của mỗi một phần động được xác định và bộ nhớ được cấp phát đủ để lưu trữ mẩu tin với phần động lớn nhất. Chẳng hạn với mẩu tin cong_nhan ta có mô hình lưu trữ như trong hình vẽ sau:

Ho_ten		Ho_ten
Ngay_cong		Ngay_cong
Luong		Luong
Loai		Loai
He_so		Gia_cong_nhat
Nghi_bhxh		Không sử dụng
Công nhân biên chế		Công nhân hợp đồng

Vì khối ô nhớ đủ lớn để lưu trữ phần động lớn nhất nên có đủ chỗ cho bất kỳ một phần động nào nhưng đối với những phần động nhỏ hơn sẽ không sử dụng tới một số ô nhớ đã được cấp phát.

Với mẩu tin có cấu trúc thay đổi, rõ ràng ta đã tiết kiệm được một số ô nhớ so với mẩu tin bình thường.

Giải thuật thực hiện phép toán

Lựa chọn một phần tử của phần động cũng giống như lựa chọn một phần tử bình thường, qua việc dịch thì độ dời của phần tử được lựa chọn sẽ được tính toán và qua việc thực hiện thì độ dời được cọng vào địa chỉ cơ sở của khối để xác định vị trí của phần tử.

CHUỖI KÝ TỰ:

Chuỗi ký tự là cấu trúc dữ liệu bao gồm một dãy các ký tự.

Như vậy, kiểu chuỗi ký tự là một kiểu đồng nhất, còn về kích thước thì có thể cố định hoặc thay đổi tùy theo ngôn ngữ. Kiểu dữ liệu chuỗi ký tự là một kiểu quan trọng mà hầu hết các ngôn ngữ đều có.

Đặc tả và cú pháp:

Đặc tả thuộc tính

Tùy ngôn ngữ, có thể có 3 cách đặc tả đối với kiểu chuỗi ký tự:

a/ Độ dài được khai báo cố định: Chuỗi ký tự có thể có độ dài (kích thước) cố định được khai báo trong chương trình. Mọi giá trị được gán cho đối tượng dữ liệu chuỗi đều có cùng độ dài như vậy. Khi một chuỗi thực được gán cho đối tượng dữ liệu mà độ dài của chuỗi thực khác độ dài được khai báo thì sẽ có sự điều chỉnh độ dài của chuỗi thực bằng cách cắt bớt các ký tự dư hoặc thêm vào các ký tự trắng để có được một chuỗi có độ dài đúng như khai báo.

Đây là kỹ thuật cơ bản được dùng trong COBOL trong đó từ khóa PICTURE được dùng để xác định số lượng ký tự, ví dụ: Last_Name PICTURE X(20) khai báo biến chuỗi ký tự Last_Name chứa một chuỗi 20 ký tự.

Trong Pascal (chuẩn) kiểu dữ liệu chuỗi ký tự không có. Thay vào đó kiểu chuỗi ký tự được biểu diễn như là một véctơ của các ký tự Last_Name: PACKED ARRAY [1..20] OF Char.

b/ Độ dài thay đổi trong một giới hạn đã được khai báo: Chuỗi ký tự có thể có độ dài cực đại được khai báo trước trong chương trình nhưng giá trị thực của đối tượng dữ liệu được lưu trữ có thể là chuỗi có độ dài ngắn hơn, thậm chí có thể là chuỗi rỗng. Trong quá trình thực hiện độ dài của giá trị chuỗi của đối tượng dữ liệu có thể thay đổi, nó cũng sẽ bị cắt nếu vượt giới hạn đã khai báo.

Đây là kỹ thuật được dùng trong PL/1 (và cả trong Turbo Pascal).

c/ Độ dài không giới hạn: Chuỗi ký tự có thể có độ dài bất kỳ và độ dài có thể thay đổi một cách động thông qua quá trình thực hiện.

Đây là kỹ thuật được dùng trong SNOBOL4.

Trong ba phương pháp nói trên thì hai phương pháp đầu cho phép cấp phát bộ nhớ cho mỗi một đối tượng dữ liệu chuỗi được xác định tại thời gian dịch. Đối vơi phương pháp thứ ba thì sử dụng cấp phát bộ nhớ động tại thời gian thực hiện. Các phương pháp khác nhau cũng đòi hỏi các phép toán khác nhau trên chuỗi. Sau đây là một số phép toán chủ yếu.

Đặc tả phép toán

Trên chuỗi ký tự, thường có các phép toán sau:

a/ Phép ghép nối (concatennation)

Ghép là phép toán nhập hai chuỗi ký tự tạo ra một chuỗi mới ví dụ nếu "//" là ký hiệu của phép ghép thì "BLOCK"//"HEAD" cho ra "BLOCKHEAD". Turbo Pascal sử dụng toán tử “+” cho phép toán ghép chuỗi.

b/ Các phép toán quan hệ trên chuỗi

Các phép toán quan hệ thông thường như bằng, nhỏ hơn, lớn hơn... trên kiểu ký tự có thể được mở rộng cho chuỗi ký tự. Tập hợp các ký tự cơ bản luôn luôn có một thứ tự. Mở rộng thứ tự này cho chuỗi ký tự thành thứ tự alphabe trong đó chuỗi A nhỏ hơn chuỗi B nếu ký tự đầu tiên của A nhỏ hơn ký tự đầu tiên của B hoặc hai ký tự đầu tiên tương ứng của chúng bằng nhau và ký tự thư hai của A nhỏ hơn ký tự thứ hai của B... Nếu chuỗi A ngắn hơn chuỗi B thì A được mở rộng bằng cách thêm vào các ký tự trắng cho dài bằng B để so sánh.

c/ Chọn chuỗi con dùng chỉ số chỉ vị trí của ký tự

Nhiều ngôn ngữ cung cấp một phép toán chọn chuỗi con của một chuỗi bằng cách cho vị trí của ký tự đầu tiên và ký tự cuối cùng của nó (hoặc vị trí của ký tự đầu tiên và chiều dài của chuỗi con). Ví dụ trong FORTRAN, lệnh NEXT = STR(6:10) là gán 5 ký tự, bắt đầu từ vị trí thứ 6 đến vị trị thứ 10 của chuỗi STR cho biến chuỗi NEXT.

d/ Định dạng nhập - xuất

Định dạng nhập xuất là phép toán dùng để thay đổi dạng nhập vào hoặc xuất ra của các chuỗi ký tự. Nhập xuất có định dạng là nét nổi bật của FORTRAN và PL/1.

e/ Chọn chuỗi con dùng so mẫu

Thông thường người ta không biết vị trí của một chuỗi con cần chọn trong một chuỗi lớn hơn nhưng quan hệ của nó với một chuỗi con khác thì có thể biết. Ví dụ chuỗi các chữ số sau dấu chấm thập phân hay chuỗi đứng sau một khoảng trống. Phép so mẫu có một đối số thứ nhất để xác định dạng của chuỗi con cần chọn (chẳng hạn độ dài của nó). Đối số thứ hai của phép toán so mẫu là chuỗi ký tự dùng để tìm trong chuỗi (chẳng hạn dấu chấm thập

phân). Như vậy kết quả của phép toán so mẫu là chọn được một chuỗi con bắt đầu từ sau dấu chấm thập phân và có độ dài đã cho.

Cài đặt

Biểu diễn bộ nhớ

Mỗi một phương pháp đặc tả chuỗi có một cách biểu diễn bộ nhớ tương ứng.

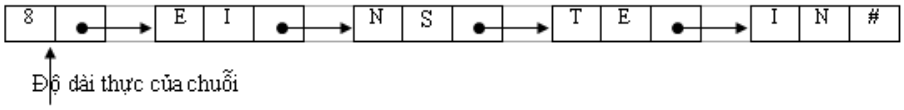
Đối với chuỗi có độ dài được khai báo cố định thì dùng véctơ của các ký tự. Ví dụ chuỗi được khai báo có độ dài 8 và được dùng để lưu trữ chuỗi EINSTEIN (cũng có 8 ký tự):

E	I	N	S	T	E	I	N

Đối với chuỗi có độ dài thay đổi trong một giới hạn đã được khai báo thì vẫn dùng véctơ của các ký tự, trong đó sử dụng hai ô làm bộ mô tả chứa giá trị thể hiện độ dài lớn nhất đã được khai báo và độ dài hiện hành của chuỗi. Ví dụ chuỗi được khai báo có độ dài 12 và được dùng để lưu trữ chuỗi EINSTEIN (có 8 ký tự):

12	8	E	I	N	S	T	E	I	N			
Độ dài khai báo	Độ dài thực								Các ô dư không sử dụng			

Đối với chuỗi có độ dài không giới hạn thì sử dụng biểu diễn bộ nhớ liên kết với bộ mô tả chứa độ dài hiện tại của chuỗi.



Giải thuật thực hiện các phép toán

Thông thường phần cứng hỗ trợ cho việc biểu diễn chuỗi có độ dài cố định nhưng đối với các biểu diễn khác cho chuỗi thì phải được mô phỏng bởi phần mềm. Các phép toán ghép, chọn chuỗi con và so mẫu phải mô phỏng bởi phần mềm.

CẤU TRÚC DỮ LIỆU CÓ KÍCH THƯỚC THAY ĐỔI

CTDL có kích thước thay đổi là một cấu trúc mà trong đó số lượng các phần tử có thể thay đổi một cách động trong quá trình thực hiện chương trình.

Một số kiểu chủ yếu của cấu trúc dữ liệu có kích thước thay đổi là:

Danh sách và cấu trúc danh sách

Danh sách là một CTDL tuyến tính với số lượng thay đổi của các phần tử có kiểu giống nhau.

Nếu các phần tử của một danh sách lại là một danh sách thì được gọi là cấu trúc danh sách (list structures).

Các phần tử có thể được thêm vào hoặc xóa khỏi một danh sách. Các phần tử có thể được lựa chọn từ một danh sách nhưng vì vị trí của phần tử trong danh sách có thể bị thay đổi do phép thêm và xóa các phần tử nên không thể sử dụng chỉ số để xác định phần tử. Thay vào đó, việc lựa chọn dựa trên cơ sở của mối quan hệ của vị trí của phần tử với danh sách chẳng hạn phần tử đầu, hai, ba, kế hặc cuối. Biểu diễn bộ nhớ liên kết cho danh sách và cấu trúc danh sách được dùng một cách phổ biến để phù hợp với sự thay đổi số lượng các phần tử.

Ngăn xếp và hàng đợi

Ngăn xếp là một danh sách mà trong đó việc lựa chọn, thêm, xóa phần tử được thực hiện ở 1 đầu của danh sách.

Do việc thêm, xóa phần tử chỉ được thực hiện ở một đầu của ngăn xếp, nên phần tử được đưa vào sau, sẽ được lấy ra trước. Do vậy ngăn xếp còn được gọi là cấu trúc dữ liệu kiểu LIFO (Last In, First Out).

Hàng đợi là một danh sách mà trong đó việc lựa chọn, và xóa phần tử được thực hiện ở một đầu còn việc thêm phần tử được thực hiện ở đầu khác của danh sách.

Do việc xóa phần tử được thực hiện ở một đầu (đầu của hàng) còn việc thêm phần tử được thực hiện ở cuối hàng, nên phần tử được đưa vào trước, sẽ được lấy ra trước. Do vậy hàng đợi còn được gọi là cấu trúc dữ liệu kiểu FIFO (First In, First Out).

Cả hai phương pháp biểu diễn tuần tự và liên kết đều được dùng cho ngăn xếp và hàng đợi.

CON TRỎ

Cấp phát tĩnh, cấp phát động và con trỏ

Cấp phát bộ nhớ (gọi tắt là cấp phát) là sự dành riêng các ô nhớ của bộ nhớ cho chương trình sử dụng.

Thông thường các ô nhớ được cấp phát để lưu trữ giá trị dữ liệu của biến. Có hai phương pháp cấp phát là cấp phát tĩnh và cấp phát động.

Cấp phát tĩnh là sự cấp phát ô nhớ cho ĐTDL được thực hiện trong quá trình dịch.

Trong khi biên dịch, thông qua sự khai báo biến, bộ dịch xác định được kiểu dữ liệu của ĐTDL nên sẽ dành sẵn một khối ô nhớ đủ lớn để lưu trữ ĐTDL của kiểu này.

Người lập trình sử dụng ô nhớ được cấp phát thông qua tên biến.

Khi khối chương trình, nơi khai báo biến kết thúc thì ô nhớ đã được cấp phát sẽ được tự động giải phóng.

Ưu điểm

Ưu điểm của cấp phát tĩnh là người lập trình dễ sử dụng, cụ thể là người lập trình chỉ cần khai báo biến, chương trình dịch sẽ tự động cấp phát và sau đó tự động giải phóng.

Nhược điểm

Nhược điểm của cấp phát tĩnh là việc sử dụng bộ nhớ không tối ưu, cụ thể là có thể cấp phát nhiều ô nhớ nhưng sử dụng không hết hoặc cấp phát thiếu.

Cấp phát động là sự cấp phát trong khi thực hiện chương trình.

Người lập trình phải viết lệnh cấp phát trong chương trình, khi thực hiện lệnh này thì bộ nhớ mới được cấp phát.

Sử dụng cấp phát động, người lập trình có thể ra lệnh giải phóng để thu hồi ô nhớ.

Để có thể cấp phát động, ta cần có một biến con trỏ hay còn gọi là biến kiểu tham chiếu. Biến con trỏ là một ĐTDL sơ cấp chứa địa chỉ của khối ô nhớ được cấp phát.

Người lập trình sử dụng ô nhớ được cấp phát thông qua biến con trỏ.

Ưu điểm

Ưu điểm nổi bật của cấp phát động là sử dụng bộ nhớ một cách tối ưu.

Nhược điểm

Nhược điểm của cấp phát động là sự lắm tên, có thể có nhiều tên biến con trỏ cùng tham chiếu đến một ô nhớ và do vậy làm giảm độ tin cậy của chương trình. Ngoài ra cũng gặp khó kăn khi sử dụng cấp phát động.

Sự đặc tả

Đặc tả thuộc tính

Có hai loại con trỏ khác nhau:

Con trỏ chỉ có thể tham chiếu tới các ĐTDL cùng kiểu

Đây là phương pháp được dùng trong Pascal và Ada.

Ví dụ trong Pascal:

Var p: ^integer chỉ ra rằng p là một biến con trỏ chứa địa chỉ của ô nhớ lưu trữ được một số integer.

Var q: ^VECT chỉ ra rằng q là một biến con trỏ chứa địa chỉ của khối ô nhớ của ĐTDL thuộc kiểu véctơ VECT nào đó.

Con trỏ có thể tham chiếu tới các ĐTDL khác kiểu nhau

Đây là cách được dùng trong các ngôn ngữ như SNOBOL4, nơi mà đối tượng dữ liệu mang bộ mô tả kiểu trong quá trình thực hiện và phép kiểm tra kiểu động được sử dụng.

Đặc tả phép toán

Các phép toán bao gồm:

Phép toán cấp phát ô nhớ động: Phép toán này dùng để cấp phát ô nhớ cho đối tượng dữ liệu mới và trả địa chỉ của ô nhớ đó về trong biến con trỏ. Đây là phép toán quan trọng nhất của kiểu con trỏ. Phép toán này có hai điểm khác biệt với việc tạo ra đối tượng dữ liệu tĩnh (bằng cách khai báo) là: Đối tượng dữ liệu được tạo ra không cần có tên vì nó được truy xuất thông qua con trỏ và đối tượng dữ liệu có thể được tạo ra một cách động trong quá trình thực hiện chương trình. Trong Pascal và Ada thì phép toán này có tên là NEW. Ví dụ NEW(p).

Phép toán truy xuất ô nhớ được cấp phát động: Để truy xuất đến giá trị dữ liệu lưu trong khối ô nhớ cấp phát động ta phải sử dụng địa chỉ của khối ô nhớ thông qua tên con trỏ (vì khối ô nhớ này không có tên). Ví dụ q^[5] là phần tử thứ 5 của véctơ Vect được trỏ bởi q.

Phép toán thu hồi ô nhớ được cấp phát động: Phép toán này cho phép giải phóng ô nhớ đã cấp phát. Trong Pascal, dùng phép toán DISPOSE.

Ví dụ sau trong Pascal minh hoạ tổng hợp các điều nói trên:

```
Type

Vect = ARRAY[1..10] of Integer;

{Lúc này bộ nhớ cho Vect chưa được cấp phát}

VAR

p: ^Vect;

{Khai báo p là một biến con trỏ chứa địa chỉ của khối ô nhớ lưu trữ ĐTDL thuộc kiểu véctơ Vect. Khi dịch đến đây thì ô nhớ cho p sẽ được cấp phát}

Begin

NEW(p);

{Cấp phát ô nhớ cho véctơ và trả địa chỉ của ô nhớ này cho biến con trỏ p (hay còn nói p trỏ tới khối ô nhớ này)}

p^[5] := 20; {Truy xuất đến phần tử thứ 5 của véctơ}

writeln(p^[5]);

Dispose(p); {Giải phóng ô nhớ đã cấp cho véctơ}

End.
```

Sự cài đặt

Có hai phương pháp biểu diễn bộ nhớ được sử dụng để biểu diễn cho một giá trị con trỏ:

Địa chỉ tuyệt đối

Giá trị con trỏ là địa chỉ ô nhớ thực của khối ô nhớ của ĐTDL.

Phương pháp này rất hiệu quả, bởi vì giá trị con trỏ tự nó quy định sự truy xuất trực tiếp tới đối tượng dữ liệu bằng cách dùng phép toán truy xuất bộ nhớ của phần cứng.

Địa chỉ tương đối

Đây là phương pháp cấp phát một vùng nhớ rộng với địa chỉ cơ sở của nó. Giá trị con trỏ là độ dời của ĐTDL. Địa chỉ của ĐTDL được tính bằng cách lấy địa chỉ cơ sở + độ dời của ĐTDL (tức là giá trị của con trỏ).

Phương pháp này thuận tiện cho việc quản lý bộ nhớ nhưng truy xuất đến ĐTDL chậm vì phải tính địa chỉ của khối ô nhớ biểu diễn cho ĐTDL.

TẬP HỢP

Đặc tả

Đặc tả thuộc tính

Tập hợp là một cấu trúc dữ liệu đồng nhất và có kích thước thay đổi.

Trong một tập hợp người ta không quan tâm đến thứ tự của các phần tử; giá trị các phần tử khác nhau.

Đặc tả phép toán

Các phép toán cơ bản trên tập hợp là:

1/ Kiểm tra sự tồn tại của một phần tử

Phép toán này dùng để xác định xem một giá trị X nào đó có phải là một phần tử của tập hợp S hay không.

2/ Thêm và bớt các phần tử cho tập hợp

Thêm giá trị X vào trong tập S, với điều kiện nó chưa là một phần tử của tập hợp. Xóa một giá trị dữ liệu X của tập S nếu nó là một phần tử của S. Hai phép toán này sẽ làm thay đổ kích thước của tập hợp.

3/ Phép hợp, giao và hiệu của 2 tập hợp

Đây là các phép toán được định nghĩa tương tự như trong toán học.

Cài đặt

Để cài đặt một tập hợp, ta có thể sử dụng một trong hai phương pháp sau:

Véctơ bit

Biểu diễn bộ nhớ

Tập hợp được biểu diễn bởi một chuỗi các bit. Cách tiếp cận này phù hợp cho một không gian nhỏ. Chẳng hạn ta có một không gian gồm n phần tử được đánh số thứ tự e1, e2, ... en. Một tập hợp các phần tử được chọn từ không gian này được biểu diễn bởi một véctơ có n bit, trong đó nếu bit thứ i có giá trị 1 thì phần tử ei thuộc vào tập hợp, ngược lại bit thứ i có giá trị 0 thì ei không thuộc tập hợp.

Giải thuật thực hiện các phép toán

Với cách biểu diễn này, việc thêm một phần tử vào trong tập hợp được thực hiện bằng cách cho bit tương ứng giá trị bằng 1. Việc xóa một phần tử trong tập hợp được thực hiện bằng cách cho bit tương ứng giá trị bằng 0. Phép kiểm tra một phần tử có thuộc tập hợp hay không được thực hiện bằng cách kiểm tra bit tương ứng có giá trị là 1 hay 0. Phép hợp của hai tập hợp tương ứng với phép toán logic OR của hai véctơ bit. Phép giao của hai tập hợp tương ứng với phép toán logic AND của hai véctơ bit. Hiệu của hai tập hợp tương ứng với phép toán logic AND của véctơ bit thứ nhất với phần bù của véctơ bit thứ hai. Các phép toán logic trên các véctơ bit đều được hỗ trợ bởi phần cứng.

Ví dụ Ta có một không gian bao gồm 5 phần tử 1,2,3,4,5. Khi đó

Tập hợp A = {1,2,4,5} được biểu diễn bởi véctơ (1,1,0,1,1)

Tập hợp B = {2,3,4} được biểu diễn bởi véctơ (0,1,1,1,0)

Do đó A B sẽ là tập {1,2,3,4,5} bởi vì (1,1,0,1,1) OR (0,1,1,1,0) = (1,1,1,1,1)

A B sẽ là tập hợp {2,4} bởi vì (1,1,0,1,1) AND (0,1,1,1,0) = (0,1,0,1,0)

A\B sẽ là tập hợp {1,5} bởi vì phần bù của (0,1,1,1,0) là (1,0,0,0,1) và

(1,1,0,1,1) AND (1,0,0,0,1) = (1,0,0,0,1)

Ưu điểm

Dễ dàng cài đặt các phép toán trên tập hợp với tốc độ thực hiện nhanh nhờ sử dụng các phép toán của phần cứng.

Nhược điểm

Không thể biểu diễn cho tập hợp mà các phần tử của nó có thể lấy từ một không gian lớn, có số lượng các phần tử bất kỳ.

Bảng băm

Biểu diễn bộ nhớ

Phương pháp này thích hợp cho các không gian lớn. Theo đó mỗi tập hợp được biểu diễn bởi một bảng băm (bảng băm mở). Mỗi phần tử của tập hợp được lưu trữ trong các lô (bucket) của bảng băm nhờ vào hàm băm (mỗi lô là một danh sách liên kết, mỗi phần tử của danh sách chứa môt phần tử của tập hợp).

Giải thuật thực hiện các phép toán

Phép toán kiểm tra sự tồn tại của một phần tử trong tập hợp được thực hiện bằng cách sử dụng phép tìm kiếm một phần tử trong bảng băm.

Các phép toán thêm và bớt một phần tử của tập hợp được thực hiện bằng cách sử dụng các phép toán tương ứng là xen và xoá một phần tử của bảng băm.

Các phép toán hợp, giao và hiệu của hai tập hợp đòi hỏi phải có một sự cài đặt công phu hơn.

Ưu điểm

Có thể biểu diễn cho tập hợp bất kỳ, không giới hạn về kích thước.Các phép toán kiểm tra một phần tử thuộc tập hợp, thêm và bớt một phần tử thực hiện dễ dàng và khá hiệu quả.

Nhược điểm

Khó khăn trong việc cài đặt các phép toán hợp, giao và hiệu của hai tập hợp.

TẬP TIN

Tập tin là một CTDL có 2 tính chất đặc biệt.

1/ Lưu trữ trong bộ nhớ ngoài như đĩa hay băng từ do đó có thể lớn hơn hầu hết các CTDL khác.

2/ Thời gian tồn tại của nó lâu dài.

Tập tin tuần tự là một kiểu phổ biến nhất của tập tin nhưng nhiều ngôn ngữ còn cung cấp tập tin truy xuất trực tiếp và tập tin tuần tự có chỉ mục.

Tập tin tuần tự

Sự đặc tả

Tập tin tuần tự là một CTDL bao gồm một dãy tuyến tính các phần tử có cùng kiểu. Độ dài của tập tin là không giới hạn. Kiểu phần tử có thể là kiểu sơ cấp hoặc kiểu cấu trúc có kích thước cố định như mảng hoặc mẩu tin. Kiểu cấu trúc có kích tước thay đổi thông thường không thể là phần tử của tập tin (do đó không có tập tin của tập tin hay tập tin của ngăn xếp).

Một cách phổ biến, tập tin có thể được truy nhập theo một trong hai mode: READ hoặc WRITE. Trong cả hai mode này đều có một con trỏ tập tin (file position pointer) dùng để xác định vị trí của phần tử nào đó hoặc sau phần tử cuối cùng. Trong mode WRITE, con trỏ tập tin luôn luôn chỉ vào sau phần tử cuối cùng và phép toán duy nhất có thể là ghi một phần tử mới vào vị trí đó. Trong mode READ, con trỏ tập tin có thể chỉ vào bất kỳ vị trí nào trong tập tin và phép toán duy nhất là đọc phần tử đó. Trong cả hai mode, phép toán READ hoặc WRITE đều di chuyển con trỏ tập tin đến phần tử kế tiếp. Nếu con trỏ tập tin chỉ tới sau phần tử cuối cùng của tập tin thì tập tin được gọi là được chỉ tới cuối tập tin (end-of-file).

Các phép toán chủ yếu đối với tập tin tuần tự là:

1/ OPEN

Thông thường một tập tin phải được mở trước khi sử dụng. Phép toán OPEN chỉ ra tên của tập tin và mode truy xuất tập tin (READ hoặc WRITE). Nếu mode là READ thì tập tin phải chắc chắn là đã tồn tại. Hệ diều hành cung cấp đặc tính của tập tin, cấp phát ô nhớ cần thiết cho vùng nhớ đệm và đặt con trỏ tập tin vào phần tử đầu tiên. Nếu mode là WRITE thì hệ điều hành tạo một tập tin rỗng, nếu tập tin đã tồn tại thì xóa tất cả các phần tử của tập tin để nó rỗng, con trỏ tập tin chỉ vào vị trí đầu tập tin rỗng.

Ví dụ trong Pascal thủ tục RESET mở một tập tin để READ và thủ tục REWRITE mở một tập tin để WRITE.

2/ READ

Phép toán READ chuyển nội dung của phần tử hiện hành của tập tin (được chỉ định bởi con trỏ tập tin) vào biến được chỉ định trong chương trình.

3/ WRITE

Phép toán WRITE tạo ra một phần tử mới của tập tin tại vị trí hiện hành và chuyển nội dung của biến chương trình được chỉ định vào phần tử mới.

4/ Kiểm tra cuối tập tin

Là phép toán xác định xem vị trí của con trỏ tập tin có nằm sau phần tử cuối cùng của tập tin hay không.

5/ CLOSE

Khi việc xử lý tập tin đã hoàn tất thì nó phải được đóng lại. Thông thường tập tin được đóng một cách tự động khi chương trình kết thúc. Tuy nhiên nếu muốn thay đổi mode truy nhập tập tin từ WRITE sang READ hoặc ngược lại thì tập tin phải được đóng một cách tường minh bằng phép toán CLOSE và sau đó mở lại cho mode mới.

Phép cài đặt

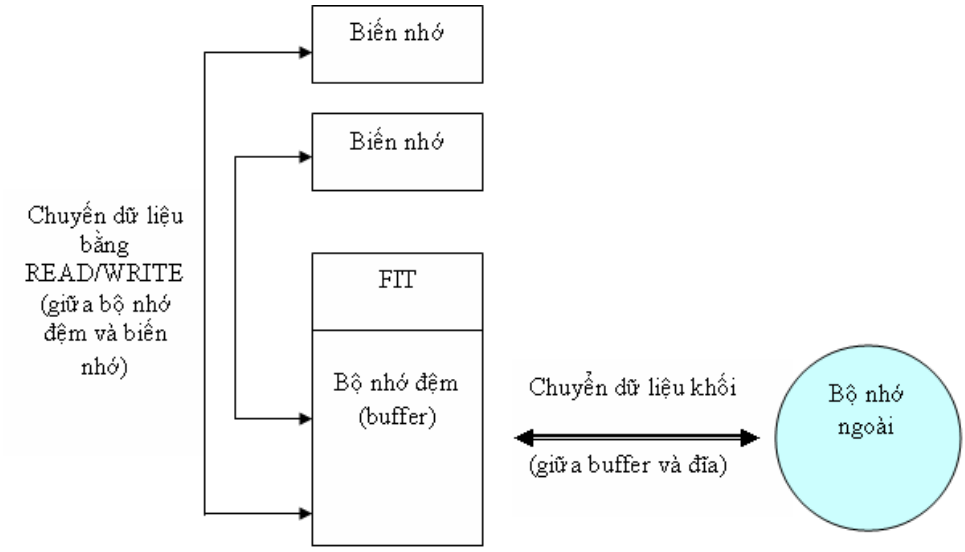
Trong hầu hết các hệ máy tính, thì hệ điều hành chịu trách nhiệm chủ yếu về việc cài đặt tập tin bởi vì tập tin được tạo ra và sử dụng bởi nhiều ngôn ngữ lập trình khác nhau. Ngôn ngữ lập trình chỉ làm một việc là cung cấp những cấu trúc dữ liệu cần thiết để giao diện với hệ điều hành.

Các phép toán trên tập tin được cài đặt một cách chủ yếu bằng cách gọi các phép toán của hệ điều hành.

Khi chương trình mở một tập tin, thì bộ nhớ lưu trữ một bảng thông tin về tập tin (FIT) (File Information Table) và một bộ nhớ đệm (buffer) được cung cấp. Phép toán OPEN của hệ diều hành sẽ lưu trữ thông tin về vị trí và các đặc tính của tập tin vào trong bảng FIT.

Nếu tập tin được mở để ghi thì khi phép toán WRITE chuyển một phần tử để nối vào cuối tập tin, thì dữ liệu được gửi cho phép toán WRITE của hệ điều hành. Phép toán WRITE của hệ điều hành sẽ lưu dữ liệu vào trong vị trí có thể của bộ nhớ đệm. Khi trong bộ nhớ đệm đã tích lũy được một khối các phần tử thì khối đó sẽ được chuyển sang bộ nhớ ngoài (đĩa hoặc băng từ). Quá trình tiếp tục của phép toán WRITE được thực hiện bằng cách lấp đầy bộ nhớ đệm cho đến khi một khối có thể được chuyển ra bộ nhớ ngoài.

Đối với READ thì ngược lại, một khối các phần tử của tập tin sẽ được chuyển sang bộ nhớ đệm và mỗi một phép toán READ được thực hiện bởi chương trình lại chuyển một phần tử từ bộ nhớ đệm sang biến chương trình cho đến khi bộ nhớ đệm trở thành rỗng thì một khối lại được chuyển từ bộ nhớ ngoài vào bộ nhớ đệm.



Tập tin văn bản

Tập tin văn bản là một tập tin của các ký tự. Đây là một loại tập tin rất thông dụng vì nó được sử dụng một cách dễ dàng trong tất cả các ngôn ngữ lập trình và các công cụ khác (Các loại tập tin khác không có được đặc điểm này). Tập tin văn bản cũng là một tập tin tuần tự nên các thao tác trên nó cũng tương tự như trên tập tin tuần tự. Ngoài ra còn có các phép toán đặc biệt khác cho phép chuyển đổi dữ liệu khác thành ký tự và ngược lại khi đọc hoặc ghi trên tập tin văn bản.

Tập tin truy xuất trực tiếp

Tập tin truy xuất trực tiếp là một tập tin được tổ chức sao cho bất kỳ một phần tử nào cũng được truy xuất một cách ngẫu nhiên. Để làm được điều đó mỗi một phần tử của nó phải có một khóa chẳng hạn khóa của mỗi phần tử là số thứ tự của nó trong tập tin. Để truy xuất phần tử bất kỳ, trước hết con trỏ của tập tin phải được di chuyển tới phần tử có khóa được chỉ định, sau đó phép toán READ hoặc WRITE mới được thực hiện. Phép toán WRITE có thể thay đổi nội dung đã có trong một phần tử đã tồn tại.

CÂU HỎI ÔN TẬP

1. Nêu định nghĩa kiểu dữ liệu có cấu trúc.
2. Nêu tên các thuộc tính của cấu trúc dữ liệu?
3. Thế nào là cấu trúc dữ liệu đồng nhất?
4. Thế nào là cấu trúc dữ liệu không đồng nhất?
5. Thế nào là cấu trúc dữ liệu có kích thước cố định?
6. Thế nào là cấu trúc dữ liệu có kích thước thay đổi?

7. Cho ví dụ về một cấu trúc dữ liệu đồng nhất.
8. Cho ví dụ về một cấu trúc dữ liệu không đồng nhất.
9. Cho ví dụ về một cấu trúc dữ có kích thước cố định.
10. Cho ví dụ về một cấu trúc dữ có kích thước không cố định.
11. Trên cấu trúc dữ liệu thường có các phép toán nào?
12. Kể tên các phương pháp lựa chọn một phần tử của cấu trúc dữ liêu?
13. Nêu tên các phương pháp biểu diễn cấu trúc dữ liệu trong bộ nhớ?
14. Phép toán lựa chọn trực tiếp (ngẫu nhiên) một phần tử của cấu trúc dữ liệu được biểu diễn tuần tự được thực hiện bằng cách nào?
15. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu có kích thước cố định?
16. Cho biết công thức xác định số phần tử của một vectơ.
17. Cho biết công thức xác định địa chỉ (vị trí) của phần tử V[i] của véctơ V.
18. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu có kích thước không cố định?
19. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu đồng nhất?
20. Có phải kiểu véctơ (mảng một chiều) là một cấu trúc dữ liệu không đồng nhất?
21. Để lưu trữ một véctơ trong bộ nhớ, người ta thường sử dụng biểu diễn tuần tự hay biểu diễn liên kết?
22. Cho biết công thức xác định số phần tử của một ma trận M[LB1..UB1, LB2..UB2] (mảng hai chiều).
23. Cho biết công thức xác định địa chỉ (vị trí) của phần tử M[i,j] của ma trận M[LB1..UB1, LB2..UB2]. Biết rằng các phần tử được lưu trữ theo trật tự dòng.
24. Cho biết công thức xác định địa chỉ (vị trí) của phần tử M[i,j] của ma trận M[LB1..UB1, LB2..UB2]. Biết rằng các phần tử được lưu trữ theo trật tự cột.
25. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo cột (trật tự cột). Hãy vẽ mô hình biểu diễn sự lưu trữ này.
26. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo dòngt (trật tự dòng). Hãy vẽ mô hình biểu diễn sự lưu trữ này.
27. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo dòng (trật tự dòng), giả sử địa chỉ cơ sở của khối ô nhớ là , kích thước bộ mô tả là D, kích thước mỗi phần tử là E. Hãy tính địa chỉ (vị trí) của phân tử A[1,2].
28. Giả sử có khai báo VAR A:array[0..3, 1..3] of integer; Các phần tử của ma trận A được lưu trữ trong bộ nhớ theo phương pháp khai triển theo cột (trật tự cột), giả sử địa chỉ cơ sở của khối ô nhớ là , kích thước bộ mô tả là D, kích thước mỗi phần tử là E. Hãy tính địa chỉ (vị trí) của phân tử A[1,2].
29. Nêu tên các thuộc tính của kiểu mẩu tin.
30. Có phải mẩu tin là một cấu trúc dữ liệu có kích thước cố định?
31. Có phải mẩu tin là một cấu trúc dữ liệu có kích thước không cố định?
32. Có phải mẩu tin là một cấu trúc dữ liệu đồng nhất?
33. Có phải mẩu tin là một cấu trúc dữ liệu không đồng nhất?
34. Để lưu trữ một mẩu tin trong bộ nhớ, người ta thường sử dụng biểu diễn tuần tự hay biểu diễn liên kết?

35. Việc lựa chọn một phần tử của mẩu tin được thực hiện bởi sự lựa chọn tuần tự hay trực tiếp?
36. Có phải mẩu tin có cấu trúc thay đổi là một cấu trúc dữ liệu có kích thước cố định?
37. Có phải mẩu tin có cấu trúc thay đổi là một cấu trúc dữ liệu có kích thước thay đổi?
38. Nêu tên các phương pháp đặc tả chuỗi ký tự.
39. Nêu tên các phép toán thường có trên kiểu chuỗi ký tự.
40. Cấp phát tĩnh được thực hiện vào lúc nào?
41. Cấp phát động được thực hiện vào lúc nào?
42. Cho biết các ưu nhược điểm của cấp phát động.
43. Sử dụng cấp phát tĩnh, người lập trình có thể chủ động giải phóng ô nhớ không?
44. Sử dụng cấp phát động, người lập trình có thể chủ động giải phóng ô nhớ không?
45. Biến con trỏ được cấp phát động hay cấp phát tĩnh?
46. Có những loại con trỏ nào?
47. Nêu tên các phép toán thường có trên tập hợp.
48. Nêu tên các phương pháp để biểu diễn một tập hợp.
49. Giả sử một tập hợp được biểu diễn bởi một vectơ bit, hãy cho biết giải thuật để thực hiện các phép toán Hợp, Giao và Hiệu hai tập hợp.
50. Sử dụng véctơ bit để biểu diễn cho một tập hợp thì có những ưu, nhược điểm gì?
51. Sử dụng bảng băm để biểu diễn cho một tập hợp thì có những ưu, nhược điểm gì?
52. Giả sử một không gian có 5 phần tử e1, e2, e3, e4, e5> Tập hợp { e2, e1, e5, e4} được biểu diễn bởi vector bit nào?
53. Giả sử có ba tập hợp A, B, C được biiểu diễn bởi ba vector bit tương ứng là (1, 0, 1, 1, 1); (1, 0, 1, 0, 1) và (1, 1, 1, 0, 1). Cho biết biểu thức liên hệ giữa các tập A,B và C?
54. Kể tên các phép toán thường có trên tập tin tuần tự.
55. Trong tập tin tuần tự, chúng ta có thể nhảy đến một phần tử bất kỳ để truy xuất nó hay không?
56. Trong tập tin truy xuất trực tiếp, chúng ta có thể nhảy đến một phần tử bất kỳ để truy xuất nó hay không?
57. Trong tập tin truy xuất trực tiếp, chúng ta có thể truy xuất các phần tử một cách tuần tự từ đầu đến cuối tập tin hay không?